UPDATING OF A RELATIONAL DATA BASE SYSTEM ON TDC-316

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

By
SIBRATAN AGARWALA

to the

COMPUTER SCIENCE PROGRAMME
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
JULY, 1978

CSP-1978-M-AGA-UPD

CERTIFICATE

This is certified that the work entitled UPDATING OF RELATIONAL DATA BASE SYSTEM ON TDC-316 is carried out under my supervision by Sri Sibratan Agarwala and has not been submitted elsewhere for a degree.

R. Sankar

R. Sankar
Professor of Computer Science
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

Kanpur
July 17, 1978

to the FOND MEMORY of my FATHER

ACKNOWLEDGEMENT

I wish to take the opportunity to record my deepest gratitude to Professor R. Sankar in whom I found a most persevering paternal figure.

I am also grateful to Messrs. B. Srinivasan, S.K. Goel and Sat Pal for the lively discussions we had on the topic and for all I was able to get from it.

Thanks are also due to Messrs. K.S. Mittal, S.P. Khattri, R.P. Jaju for timely maintenance of the computer, to Mr. H.K. Nathani for his elegant typing and to all staff of the Computer Centre for the co-operation I received from them.

- Sibratan Agarwala

Kanpur
July 17, 1978

ABSTRACT

In this thesis, we have considered the problem of updating a data base system based on relational model. Data base is implemented on TDC-316 computer available in the Computer Centre, Indian Institute of Technology, Kanpur. This forms a part of a larger project involving building, retrieval, and security of data base.

In updating we have considered the following three operations -

(1) Insertion
(2) Modification
(3) Deletion.

Each of these three operations can be performed on a single relation or the whole data base. Algorithms for these operations have been designed and implemented in Basic Assembly Language.

TABLE OF CONTENTS

1. INTRODUCTION

Information updating system works on a personnel data base developed at Indian Institute of Technology, Kanpur as an M.Tech. project on TDC-316 computer. A personnel data base is a data-base which stores information of persons and in our case persons are army officers. Whole data base consists of 80 fields and 12 relations, each relation being in third normal form.

The values of different fields in a personnel data base are subject to changes from time to time. Moreover a situation may arise when new records are to be added to or records already existing are to be deleted from a single relation on the whole data base. These types of operations are called updating. Thus updating consists of three operations, namely -

(1) Deletion
(2) Modification
(3) Insertion

Deletion

In this operation, all the informations of user specified army officer(s) are deleted from a single relation or from the whole data base.

Modification

In this operation, values of some fields of some persons, specified by user are modified in a single relation or whole data base without creating any new records.

Insertion

In this operation, informations of some new persons, specified by user are inserted in a single relation on whole data base in the form of new records.

When updating is based on a single relation we call it "single relation updating" and when it is based on all the relations present in data base we call it "group relation updating."

To illustrate updating operation let us consider a data base having two relations as follows:

Relation R1

Personnel No.	Rank	Qualification
IC-00005	CAPT	B.Sc.
IC-00006	MAJOR	B.A.
IC-00010	CAPT	B.E.

Relation R2

Personnel No.	Languages Known	Service Experience
IC-00005	Hindi	5
IC-00005	Bengali	5
IC-00006	Tamil	10
IC-00010	Telugu	8
IC-00010	Malayalam	8
IC-00010	Panjabi	8

1. Delete

(a) Group Relation Deletion: Let us assume that an army officer having personnel No. IC-00010 resigns from army and all his informations, therefore, are to be deleted from data base. These relations after deletion will look like as follows:

Relation R1

Personnel No.	Rank	Qualifi-cations
IC-00005	CAPT	B.Sc.
IC-00006	MAJOR	B.A.

Relation R2

Personnel No.	Languagues Known	Service Experience
IC-00005	Hindi	5
IC-00005	Bengali	5
IC-00006	Tamil	10

(ii) Single Relation Deletion: If the informations of personnel No. IC-00010 is deleted from only relation 2 data base after deletion will look like as follows:

Relation R1

Personnel No.	Rank	Qualifi-cations
IC-00005	CAPT	B.Sc.
IC-00006	MAJOR	B.A.
IC-00010	CAPT	B.E.

Relation R2

Personnel No.	Languages Known	Service Experience
IC-00005	Hindi	5
IC-00006	Bengali	5
IC-00006	Tamil	10

2. Modification

(i) Group Relation Modification: If it is observed that rank, qualification, service experience of personnel No. IC-00006 needs modification, the operation involved is group relation modification because the fields to be modified exist in more than one relation. If the rank becomes captain, qualification becomes M.A., and service experience becomes 12 years and the relations will look like -

Relation R1

Personnel No.	Rank	Qualifica-tions
IC-00005	CAPT	B.Sc.
IC-00006	MAPT	M.A.
IC-00010	CAPT	B.E.

Relation R2

Personnel No.	Languages Known	Service Experience
IC-00005	Hindi	5
IC-00005	Bengali	5
IC-00006	Tamil	12
IC-00010	Telugu	8
IC-00010	Malayalam	8
IC-00010	Punjabi	8

(ii) Single Relation Modification: If the fields to be modified exist in a single relation the operation involved is single relation modification. If only rank and qualification of Personnel No. IC-00006 need to be modified, relation R2 in the above case will remain unaltered.

3. Insertion

(i) Group Relation Insertion: If information of a new person having Personnel No. IC-00015, say, is to be inserted relations get modified to:

Relation R1

Personnel No.	Rank	Qualifi-cations
IC-00005	CAPT	B.Sc.
IC-00006	MAJOR	B.A.
IC-00010	CAPT.	B.E.
IC-00015	MAJOR	M.Sc.

Relation R2

Personnel No.	Languages Known	Service Experience
IC-00005	Hindi	5
IC-00005	Bengali	5
IC-00006	Tamil	10
IC-00010	Telugu	8
IC-00010	Malayalam	8
IC-00010	Punjabi	8
IC-00015	Kashmiri	0

This is group relation insertion because records have been inserted in more than one relation.

(ii) Single Relation Insertion: If Personnel No. IC-00006 learns one more language only Relation 2 gets modified to -

Personnel No.	Languagues Known	Service Experience
IC-00005	Hindi	5
IC-00005	Bengali	5
IC-00006	Tamil	10
IC-00006	Malayalam	10
IC-00010	Telugu	8
IC-00010	Malayalam	8
IC-00010	Punjabi	8

This is single relation insertion.

Special Cases -

(i) If value of a field for any personnel number is to be deleted the operation involved is modification. For example, if we want to delete value of field "RANK" for Personnel No. IC-00005 we have to modify this record by putting blanks as value of this field. Relation R1 after this modification takes the form -

Personnel No.	Rank	Qualifications
IC-00005	⁄⁄---⁄⁄	B.Sc.
IC-00006	MAJOR	B.A.
IC-00010	CAPT	B.E.

(ii) If a field is such that it can have more than one value for the same personnel number (for example, "Languages Known") and if it is to be modified the operations involved are deletion and insertion. For example, if personnel number IC-00005 knows language URDU instead of HINDI then all the records for that personnel number are to be deleted from that relation first and then new records are to be inserted. After this operation R2 takes the form -

1st Stage (after deletion

Personnel No.	Languages Known	Service Experience
IC-00005	⁄⁄ --- ⁄⁄	5
IC-00005	⁄⁄ --- ⁄⁄	5
IC-00006	Tamil	10
IC-00010	Telugu	8
IC-00010	Malayalam	8
IC-00010	Punjabi	8

2nd Stage (After insertion)

Personnel No.	Languages Known	Service Experience
IC-00005	Urdu	5
IC-00005	Bengali	5
IC-00006	Tamil	10
IC-00010	Telugu	8
IC-00010	Malayalam	8
IC-00010	Punjabi	8

This operation will be called as special-modification operation.

2. DESCRIPTION OF DATA STRUCTURES

Data structures for updating operation are built based on the following assumptions:

(i) Out of three operations involved in updating, namely insertion, modification and deletion only one can be performed at a time except in the case of special-modification operation.

(ii) Any No. of fields can be updated at a time depending on the size of data structures. Size has been allotted such that a maximum of 40 fields can be updated at a time.

(iii) Any number of person's informations can be updated at a time depending on the size of a data structure USFLTB to be explained latter.

(iv) Updating is performed either on a single relation or a group of all the relations having the same primary key. For example, if relations 3,5 and 9 are having same primary key, updating of relations 3 and 5 but not 9 is not possible at a time. Relations 3 and 5 have to be updated separately as single relation updating. This type of assumption is quite reasonable due to the fact that when some field is updated, our aim will be to update all those relations in the whole data base wherever that field occurs. Updating some relations and not the rest may create discrepancy in data base.

(v) If user specifies some fields for updating all of which are not present in a relation, processing will not be deleted but all those fields which are present will be updated.

(vi) In case of single relation updating user has to specify the relation but not in group relation updating.

For updating operation user has to specify a file called user file. A sample user file is given in Figure 1

Primary Key	Fields to be updated		
Personnel No.	Rank	Salary	Qualification
IC-00005	CAPT.	1500	*
IC-00008	*	*	M.Tech.

Figure 1

Each column of this file represents a field to be updated except the primary key. Each row is a record of user file. A special character as a value of a field against a personnel number indicates that this field need not be updated for that person.

In single relation updating, all the fields specified in user's file will be updated only in the specified relation. In group relation updating all the fields specified in user's file will be updated in all those relations whose primary key is the same as the primary key of user's file.

In case of deletion user's file consists of only primary key.

All thefields of user's file are specified by user according to the following format:

LEVEL ß FIELD NAME ß FIELD CODE ß FORMAT.

LEVEL = 22 for primary key
= 44 for filler field
= 02 for ordinary field
= 0 to indicate end of field specification.

Field can be specified either through keyboard or card reader.

Example

```
22 ß PNUM ß F1 ß X*8.
44 ß FILLER ß X*4.
02 ß RANK ß F3 ß X*6.
44 ß FILLER ß X*4.
02 ß SALARY ß F9 ß 9*6.
44 ß FILLER ß X*6.
02 ß QUALF ß F25 ß X*8.
00.
```

Figure 2

Informations of above user's file fields are stored in the following data structures:

Data Structure FLD:

It stores alphabetic part of field code in one byte. Size of the array is 40 bytes.

Data Structure INFLCD:

It stores numeric part of field code in one byte. Size of the array is 40 bytes.

Data Structure INFLGT:

It stores length of field in one byte. Size of the array is 40 bytes.

Data Structure DIST:

It stores distance of this field from the beginning of a user file record in one byte. Size of array is 40 bytes.

Data Structure LDIST:

It stores distance of a field from the beginning of a user file record after removing fillers in one byte. Size of array is 40 bytes.

Data Structure FORMA:

It stores format code of the fields specified in user's file in one word. Size of array is 40 words.

Example

When fields are specified as in Figure 2 contents of different data structures will be as follows -

FLD	INFLCD	INFLGT	DIST	LDIST	FORMA
F	1	8.	0.	0.	030010
F	3	6.	12.	8.	030006
F	9	6.	22.	14.	000006
F	25	8.	34.	20.	030010

Figure 3

These data structures are printed in keyboard for user verification.

Data Structure USFLTB (User File Table):

This data structure stores values of fields specified in user file after removing fillers. Each byte stores one character and size of USFLTB is octal 4000 bytes. If average length of a record in user's file is 40 bytes informations of 50 persons can be updated at a time.

Field values are specified either through keyboard or card reader depending on user's choice. Each record must be embedded with fillers as specified in field specification of user's file. These values are stored in USFLTB after removing fillers. Necessity of storing values in some area in main memory arises due to the fact that more than one relation may haveto be updated in case of group relation updating. An example of how to specify values of fields corresponding to Figures 1 and 2 is as follows -

Record No. 1

IC-00005ØØØØCAPT.ØØØØØ 1500ØØ ØØØØØØ *ØØØØØØØ

Record No. 2

IC-00008 ØØØØ *ØØØØØ ØØØØ *ØØØØØ ØØØØØØ M.Tech.Ø

When these records are specified through keyboard each record must be terminated by a carriage return and when these records are specified by punched cards each record must start from a fresh card and one record may be punched in 5 cards at maximum, maintaining the correct size of fillers. Each record after checking the correct presence of fillers and correct record length is stored in USFLTB after removing fillers. An example of how the above two records will be stored in USFLTB is

as follows -

IC-00005CAPT.Ø1500ØØ*ØØØØØØØIC-00008*ØØØØØ*ØØØØØM.Tech.Ø

So far, the data structures necessary for updating operation have been described. These data structures are built before the actual execution of updating operation starts. Now other data structures which are built during the execution phase will be described.

Data Structure FDFLAG

Before execution of updating operation starts, it must be checked whether the fields specified in user file are present in a relation. In case of single relation updating, it is expected that all the fields specified in user's file are present in the specified relation to be updated. If it is not, it must be indicated by flags which fields are common between user specified file and the specified relation and only those fields will be updated. This is valid for group relation updating also;

In this case, all the fields which are common between user specified file and a relation are updated. When some field is found to be common, corresponding byte in FDFLAG data structure is set to 1, else it is cleared. When there is no field common this relation is skipped.

To check which fields are common between user specified file and a relation is done with the help of a data structure FDLIST (field list). FDLIST is constructed while building the data base. It stores the following informations:

(i) Field codes of fields present in a relation

(ii) Distance of a field from the start of a record in a relation.

(iii) Format code of field.

Size of FDFLAG is 40 bytes. One byte is reserved for one field.

Data Structure CMFLDS (Common Field Distance)

This data structure stores distance of a field from the start of a record in a relation, for those fields whose FDFLAG byte is set. When a field is found to be common between user specified file and a relation, the distance of the field from the start of a record in that relation is read from FDLIST and is stored in the corresponding byte of CMFLDS. Size of CMFLDS data structure is 40 bytes.

Example

Let us assume FDLIST contains the following informations:

Rel. No.	Field code	Distance	Format
1	1	0	030010
1	5	8	020005
1	19	14	010102
⋮			⋮
4	1	0	030006
4	3	8	030006
4	25	14	03001C
⋮			⋮
6	1	0	030010
6	25	8	030010
6	9	16	030006
⋮			⋮

Assuming the user file as in Figure 1 with specified fields as in Figure 2, contents of different data structures will be as follows:

While Updating Rel. 1

INFLCD	FDFLAG	CMFLDS
1	1	0
3	0	-
9	0	-
25	0	-

Wuile Updating Rel. 4

INFLCD	FDFLAG	CMFLDS
1	1	0
3	1	8
9	0	-
25	1	14

While Updating Rel. 6

INFLCD	FDFLAG	CMFLDS
1	1	0
3	0	-
9	1	16
25	1	8

Following data structures, constructed by BUILD subroutine, while building the relations, are repeatedly used in updating operation. Hence a brief description of these data structures are given below.

Relation Directory (RELDIR):

This data structure stores various informations about a relation and is stored columnwise in memory. A sample relation directory is shown below in a tabular form. N-th row of this table corresponds to Relation No. N.

RELNUM	RELID	PMINX	NIND	TOTREC	KEYRP	FMKRP	CRCNO	LNGTRC
0	0	0	0	0	0	0	0	0
NAME	R1	27000	3	1000	F1	30006	0	26
QUALF	R2	27040	2	500	F1	30006	0	12

Various columns of this table have the following interpretations:

RELNUM : It stores 8 character long name given to a relation.

RELID : It stores internal identification number given to a relation.

PMINX : It stores the starting address of Primary Index Table stored in memory corresponding to a relation.

NIND : It stores number of records in Primary Index table of a relation.

TOTREC : It stores total number of records stored in a relation.

KEYRP : It stores internal identification number of primary key of a relation.

FMKRP : It stores format of primary key.

CRCNO : It stores overflow cylinder number of a relation. This is to be constructed only by insertion operation.

LNGTRC : It stores record-length of a relation.

Primary Index Table

Corresponding to each relation one primary index table is maintained. This table stores informations of the type, which are the cylinders used to store the relation, highest primary key value in each cylinder and number of records in Secondary Index Table corresponding to each cylinder. This table is stored row-wise in core memory. A sample primary index table is as follows:

Highest key in the cylinder	Cylinder No.	No. of records in Secondary Index Table
IC-00010	13	100
IC-00300	20	100
IC-00350	11	44

Secondary Index Table

For each cylinder, one secondary index table is maintained in its 0th surface. This table stores informations of the type, which sectors in the cylinder are used to store records, highest primary key value in a sector and number of records stored in it. This table is stored row-wise. A sample secondary index table is as follows:

No. of records in sectors	Highest Primary Key in sectors	Address of sectors
10	IC-00013	41
8	IC-00025	42
0		43

Overflow Area

Overflow area is a collection of a few cylinders on disk. The necessity of overflow area arises in insertion operation. While inserting a new record, it may happen that according to primary and secondary index tables either no sectors, no cylinder is allotted to that record or some sectors, some cylinder is allotted but no space is actually available in that sector, when no space is allotted, naturally record has to be stored in the last sector or a fresh

sector on even a fresh cylinder, if necessary. But if a sector is allotted, it has to be stored only in that sector to maintain the characteristics of Data Base that records are sorted before being stored. If space is available in the allotted sector there is no problem but if it is not available there are two alternatives. First alternative is to move a few records from this sector to next sector and make space available. But this process is very time consuming and also tedious due to the fact that the immediate next sector may not have space available in it. In that case, record has to be moved from one sector to the next sector until some sector is found where some space is available, Corresponding secondary index table will have to be changed drastically. Moreover, if it is observed that no space is available in the whole cylinder then transfer of records from one cylinder to another will be necessary until a cylinder is found where space is available. This will cause changes in primary index table also. If no such cylinder is found the record is to be stored in a fresh cylinder and primary index table will have an extra entry corresponding to this cylinder.

Second alternative is that when it is found for the first time that record does not get enough space in the allotted sector, an empty cylinder is found out and it is allotted to the corresponding relation as an overflow area. Thus each relation will have at most one cylinder as overflow area and this overflow cylinder will be allotted to the relation only when such an overflow situation takes plac

As soon as, an overflow cylinder is allotted to a relation this cylinder number will be stored in corresponding entry of this relation in CRONO data structure of relation directory. In first two bytes of first sector of overflow cylinder highest sector number of overflow cylinder is stored which stores records. Records are stored in overflow cylinder starting from second sector. When an overflow occurs a free sector of overflow cylinder is foundout and its surface-sector number is stored in last 2 bytes of the original sector from which overflow has occurred. For subsequent overflows in this sector, this will act as a pointer to the overflow sector. In this scheme, corresponding to each sector in original cylinder one sector is allotted as an overflow sector. Thus during insertion operation if overflow occurs from more than 99 sectors of a relation, operation stops and DBA has to re-arrange his DATA BASE before anymore records can be inserted. On the other hand, if there is an overflow in the overflow sector itself, next available sector in overflow cylinder will be allotted as overflow area to this overflow sector and process becomes recursive.

An important point to note is that we are assuming that last 2 bytes of each sector are free so that a pointer can be stored here. This is consistent with BUILD subroutine which builds the whole data base.

-

3. CONSTRUCTION OF DATA STRUCTURES

Construction of data structures is the initial phase of updating operation. In this chapter, its algorithm is given. Various subroutines needed for this purpose are briefly described at the end of this chapter.

Algorithm

Step 1: Specify operation.

Step 2: Is it updating? If yes go to Step 6 else go to Step 3.

Step 3: Is it retrieval? If yes go to Step 4 else go to Step 5.

Step 4: Stop.

Step 5: Is it DBA operation? If yes go to Step 4 else go to Step 1.

Step 6: Give input maximum for the fields of user's file.

Step 7: Is input medium card-reader? If yes go to Step 9 else go to Step 8.

Step 8: Is input medium keyboard? If yes go to Step 9 else go to Step 6.

Step 9: Specify user file fields through specified input medium with primary key as first field and scan it.

Step 10: Is LEVEL = 0? If yes go to Step 23 else go to Step 11.

Step 11: Is LEVEL = 22? If yes go to Step 12 else go to Step 14.

Step 12: Is primary key already specified? If yes go to Step 13 else go to Step 19.

Step 13: Give message "Primary Key is Specified more than once, field ignored". Go to Step 9.

Step 14: Is primary key already specified? If yes go to Step 16 else go to Step 15.

Step 15: Give message "Primary key not yet specified". Go to Step 9.

Step 16: Is LEVEL = 44? If yes go to Step 22 else go to Step 17.

Step 17: Is LEVEL = 2? If yes go to Step 19 elsego to Step 18.

Step 18: Give message "Illegal level number, field ignored". Go to Step 9.

Step 19: Transfer alphabetic part of field code to FLD, numeric part to INFLCD, format to FORMA.

Step 20: Find out length of field and transfer it to INFLGT. Transfer actual distance of field to DIST and distance after removing fillers to LDIST.

Step 21: Update total length of user-file-record. Go to Step 9.

Step 22: Find out length of filler field. Go to Step 21.

Step 23: Total number of fields specified = 0? If yes go to Step 9 else go to Step 24.

Step 24: Print contents of data structures FLD, INFLCD, INFLGT, DIST, LDIST and FORMA. Check whether there is any field specified such that it does not exist in FDLIST or any field duplication or wrong format specification. If yes set ERRFLG. Go to Step 25.

Step 25: ERRFLG is set? If yes go to Step 9, else go to Step 26.

Step 26: Specify input medium for specifying records of user's file.

Step 27: Specify a record of user'sfile through specified input medium.

Step 28: Check whether record length of specified record is correct. If yes go to Step 30, else go to Step 29.

Step 29: Give message "Record length incorrect. Specify this record again". Go to Step 27.

Step 30: Check whether fillers are properly placed. If yes go to Step 32, else go to Step 31.

Step 31: Give message "Fillers are not properly placed". Specify this record. again. Go to Step 27.

Step 32: Store the record in USFLTB after removing fillers. Increment total number of records in user's file by 1.

Step 33: End of user's file? If yes jump to processing phase, else go to Step 27.

End of Algorithm.

To implement the above algorithm following subroutines have been written:

(1) Subroutine DCONV (decimal conversion) -

This subroutines transfers contents of a word to a buffer of size 6 bytes to print the content in decimal notation.

Example -

```
        JMS  R2,  DCONV
        WORD NUM
        WORD BUF+6

NUM : WORD 50.
BUF : WORD 6,0,6
    . = .+6
```

(2) Subroutine OCONV (Octal conversion) -

This subroutine transfers contents of a word to a buffer of size 6 bytes to print the content in octal notation.

Example -

```
        JMS R2, OCONV
        WORD NUM
        WORD BUF+6

NUM :  WORD 50.
BUF :  WORD 6,0,6
    .  = .+6
```

(3) Subroutine FLCHEK (field check)

This subroutine checks validity of field codes specified in user file. Field codes must have F as alphabetic part and non-zero positive integer not more than 80 as numeric part. If there is any field code duplication their formats must be different because some field may

occur in 2 relations in two different formats. If any of the above mentioned conditions are violated for any field specified in user's file ERRFLG flag is set. Calling instruction, JMS R3, FLCHEK.

(4) Subroutine FMCHEK (format check) -

This subroutine checks whether a field specified in user's file is present in FDLIST with the specified format. If not, it sets a flag ERRFLG.

Calling instruction, JMS R3, FMCHEK.

(5) Subroutine FMLNGT (format length)

This subroutine accepts format code of a field in octal as input and outputs its length.

Example:

```
        JMS R5, FMLNGT
        WORD FORMAT
        WORD LENGTH
FORMAT : WORD 030010
LENGTH : .=.+2
```

(6) Subroutine PRUSFUL (Print user file)

This subroutine prints the contents of different data structures FLD, INFLCD, INFLGT, DIST, LDIST and FORMA for manual verification calling instruction, JMS R3, PRUSFUL.

(7) Subroutine BLANKB (blank buffer)

This subroutine puts blanks in a buffer -

Example:

```
        JMS R2, BLANKB
        WORD BUF+6
        WORD SIZE
BUF  : WORD 100., 0, 100.
     . = .+100.
SIZE : WORD 50.
```

Contents of first 50. bytes of BUF starting from BUF+6 will be blanks.

(8) Subroutine CHEKBL (check blank)

It checks whether contents of a certain area are blank or not. if not it sets flag ERFLG1.

Example:

```
        JMS R2, CHECKBL
        WORD BUF+6
        WORD SIZE
SIZE  : WORD 2
 BUF  : WORD 100,0,100
      . = .+100
```

If contents of BUF+6 and BUF+7 are blanks ERFLG1 is reset else it is set.

(9) Subroutine FILCHK (filler check)

This subroutine checks whether fillers are properly placed in a user file record. If not, it sets a flag ERFLG1. Calling instruction JMS R6, FILCHK.

(10) Subroutine STORE

This subroutine stores user's file record from a buffer to USFLTB after removing fillers -

Example -

```
        TSR #USFLTB, LOC
        JMS R6, STORE
        WORD LOC
 LOC  : WORD 0
```

(11) Subroutine DATAIN (Data Input)

Calling instruction, JMS R5, DATAIN.

This subroutine accepts user's file records. It calls following two subroutines.

(12) Subroutine KBDIN (Input through keyboard)

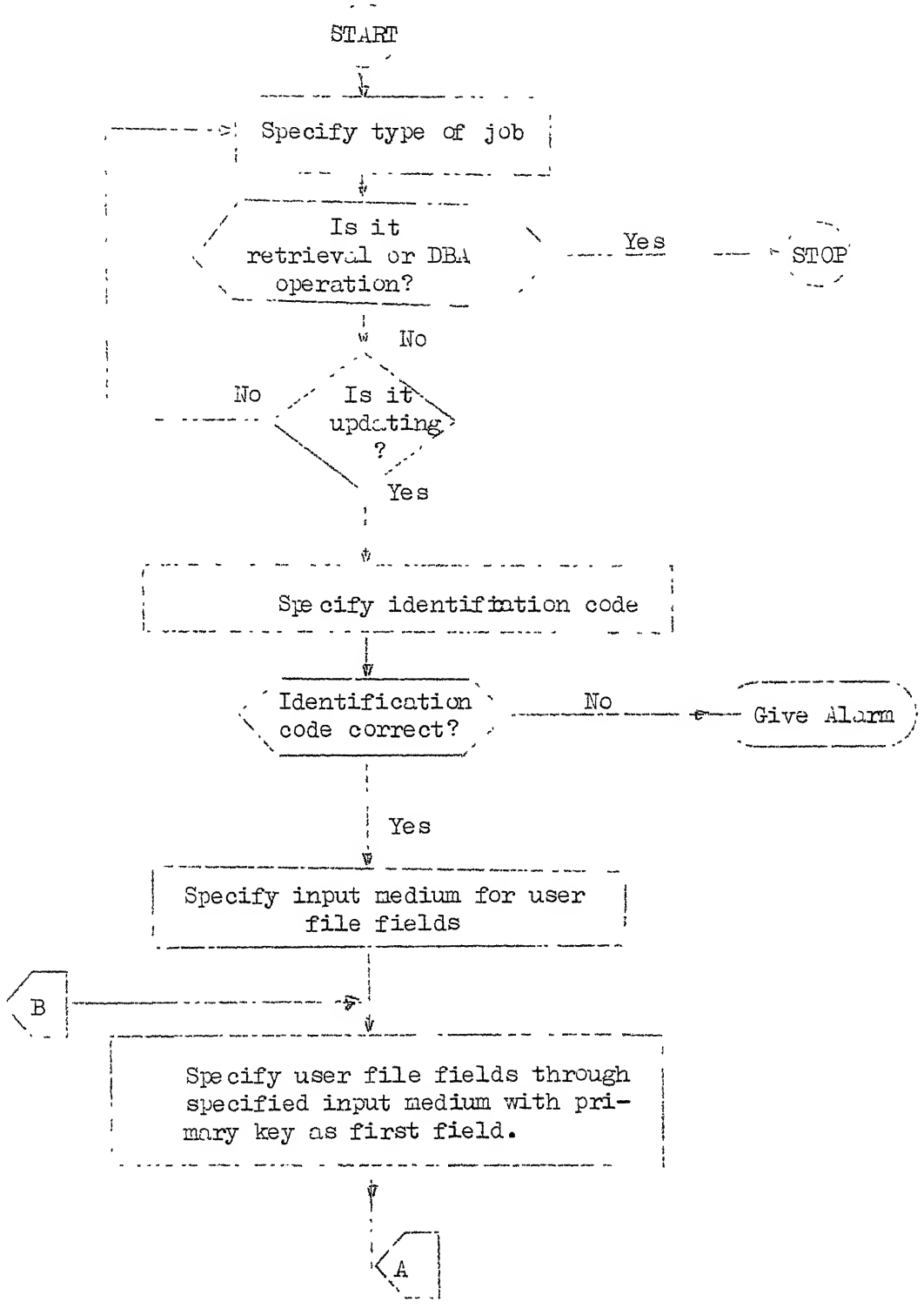
It accepts user's file records through keyboard calling instruction:

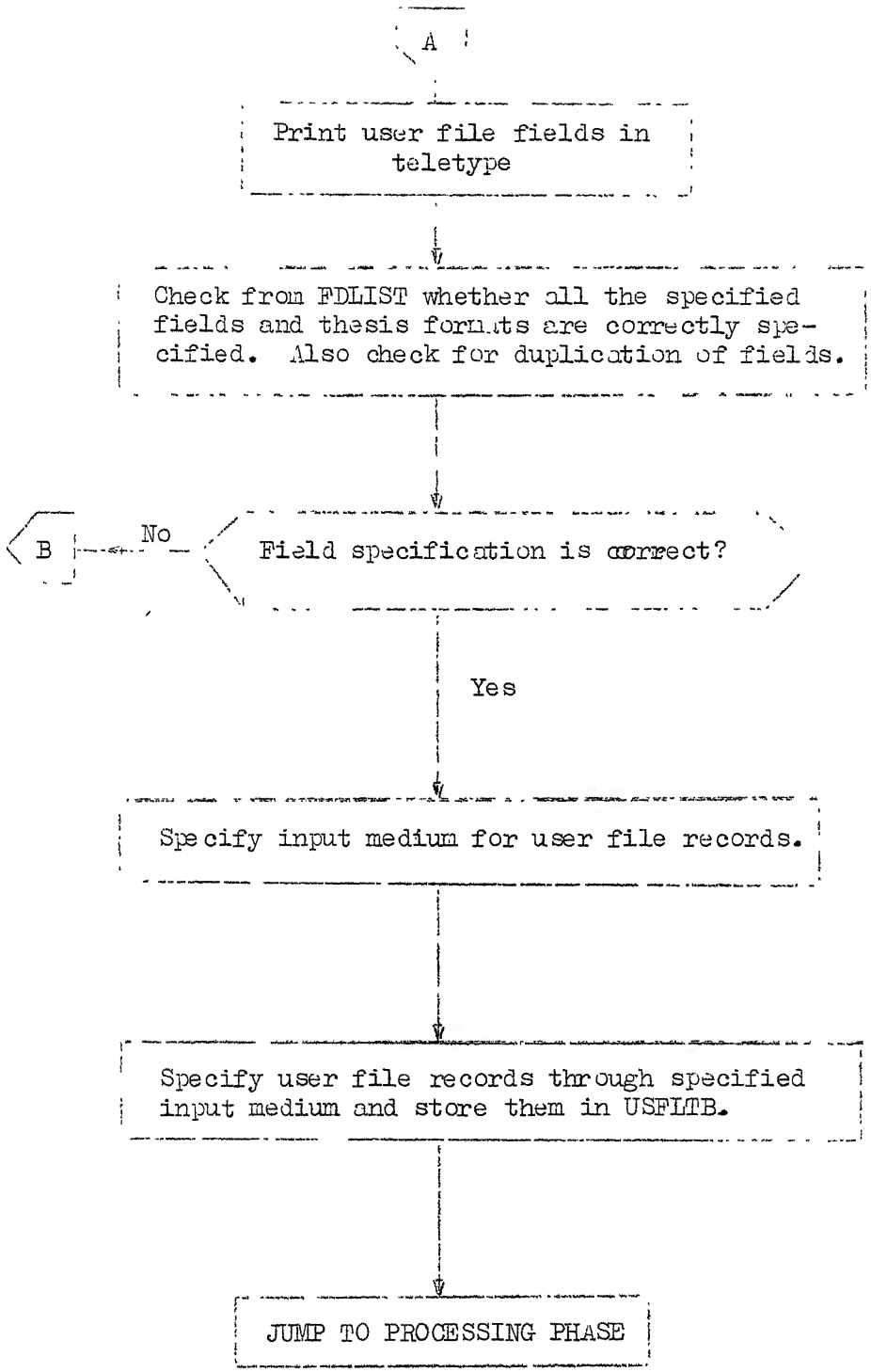
JMS R3, KBDIN.

(13) Subroutine CARDIN (Input through card-reader)

This subroutine accepts user file records through card-reader. Calling instruction JMS R3, CARDIN.

Flowchart Phase-I




A
Print user file fields in teletype
Check from FDLIST whether all the specified fields and thesis formats are correctly specified. Also check for duplication of fields.
Field specification is correct?
No
B
Yes
Specify input medium for user file records.
Specify user file records through specified input medium and store them in USFLTB.
JUMP TO PROCESSING PHASE

4. PROCESSING PHASE

When all the data structures have been built, processing of actual updating operation starts. Here user has to specify what type of updating operation it is, whether insertion, deletion or modification. Also he has to specify whether updating is based on a single relation or group of relations. If it is single relation, it must be checked from FDLIST whether the primary key and its format are correctly specified and other fields specified in user file do at all exist in the specified relation. FIFLAG and CMFLDS data structures are correspondingly built. Various informations regarding the specified relation are read from relation directory. Delete, Insert or Modify subroutine is called depending on whether the updating operation is deletion, insertion or modification respectively.

On the other hand, if updating is based on group of relations, all the relations are found out from relation directory whose primary key is the same as primary key of user file and the process of single relation updating is repeated for all these relations.

Algorithm for processing phase is as follows -

Step 1: Calculate primary and secondary indextable record lengths. Initialize BSLOCP to the starting address of USFLTB. Specify whether updating operation is insertion, deletion or modification.

Step 2: Is updating operation Insertion? If yes go to Step 5 else go to Step 3.

S

Step 3: Is updating operation deletion? If yes go to Step 5 else go to Step 4.

Step 4: Is updating operation modifcation? If yes go to Step 5 else go to Step 1.

Step 5: Specify whether updating is based on a single relation or group of relations.

Step 6: Is updating based on a single relation? If yes go to Step 8, else go to Step 7.

Step 7: Is updating based on group of relations? If yes go to Step 35, else go to Step 5.

Step 8: Specify relation no. and store it in RELCD.

Step 9: Is specified relation no. RELCD less than 1 or greater than 39? If yes go to Step 10, else go to Step 11.

Step 10: Give message "Relation No. wrongly specified, specify it again". Go to Step 8.

Step 11: Check from relation directory whether relation RELCD exists in Data Base. If yes go to Step 13, else go to Step 12.

Step 12: Give message, "Specified relation does not exist in Data Base, specify it again.". Go to Step 8.

Step 13: From relation directory check whether primary key of user file is correctly specified. If yes, go to Step 15, else go to Step 14.

Step 14: Give message "Primary key of user file does not matdh." Go to Step 8.

Step 15: From relation directory check whether format of primary key of user file is correctly specified. If yes go to step 17, else go to Step 16.

Step 16: Give message, "Format of Primary Key does not match". Go to Step 8.

Step 17: Check whether relation RELCD exists in FDLIST. If yes go to Step 19, else go to Step 18.

Step 18: Give message, "Relation does not exist in FDLIST. Relation directory and FDLIST are inconsistent." Go to Step 5.

Step 19: Check from FDLIST whether there is any field common between user file and the specified relation. If yes, go to Step 21, else go to Step 20.

Step 20: Give message, "No common field between user file and the specified relation. Inconsistency between FDLIST and relation directory." Go to Step 5.

Step 21: Construct data structures FDFLAG and CMFLDS. Is No. of common field = 1? If yes go to Step 22, else go to Step 24.

Step 22: Is the updating operation deletion? If yes go to Step 27, else go to Step 23.

Step 23: Give message, "Only primary key common between user file and specified relation." Go to Step 5.

Step 24: Is No. of common field = total No. of fields in user file? If yes go to Step 27, else go to Step 25.

Step 25: Give message, "All the fields specified in user file are not present in specified relation. Shall I proceed?"

Step 26: If reply = yes, go to Step 27, else if reply = No go to Step 5, else go to Step 25.

Step 27: Read various informations about relation RELCD from relation directory and calculate MAXRC, maximum no. of records in a sector.

Step 28: Is updating operation deletion? If yes go to Step 29, else go to Step 30.

Step 29: Call subroutine DELETE. Go to Step 33.

Step 30: Is updating operation insertion? If yes go to Step 31, else go to Step 32.

Step 31: Call subroutine INSERT. Go to Step 33.

Step 32: Call subroutine MODIFY. Go to next step.

Step 33: Updating is based on single relation? If yes goto Step 34, else go to Step 36.

Step 34: Is operation deletion? If yes go to Step 46, else stop.

Step 35: Assume relation No. = 0 and store it in RELCD.

Step 36: Increment RELCD by 1.

Step 37: Is RELCD 39. If yes, go to Step 34, else go to Step38.

Step 38: Check from relation directory whether relation RELCD exists in Data Base. If yes go to Step 39, else go to Step 76.

Step 39: From relation directory check whether primary key of user file matches. If yes go to Step 40, else go to Step 36.

Step 40: From relation directory check whether format of primary key of user file matches. If yes, go to Step 41, else go to Step 36.

Step 41: Check whether the relation exists in FDLIST. If yes go to Step 43, else go to Step 42.

Step 42: Give message, "Relation directory and FDLIST are inconsistant" Go to Step 36.

Step 43: Check from FDLIST whether there is any field common between user file and fields present in this relation. If yes go to Step 44, else go to Step 42.

Step 44: Construct data structures FDFLAG and CMFLDS. Is No. of common field = 1? If yes, go to Step 45, else go to Step 27.

Step 45: Is updating operation deletion relation? If yes, go to Step 27, else go to Step 36.

Step 46: Is operation special modification? If yes go to Step 6 of Phase I algorithm, else Stop.

End of Algorithm.

A brief account of different subroutines written to implement this algorithm is given below:

(1) Subroutine COMRLD (Compare Relation Directory)

This subroutine compares primary key of user file and its format with corresponding entries in relation directory for a given relation No. RELCD. If primary key matches it sets a flag PKFLAG and if its format also matches it sets a flag FMFLAG, else both flags are reset. Calling instruction is JMS R2, COMRLD.

(2) Subroutine READRD (Read Relation Director)

This subroutine reads various entries in relation directory corresponding to relation No. RELCD. Calling instruction is JMS R2, READRD.

(3) Subroutine CONVRP (Convert Reply)

This subroutine converts a number stored in ASCII code in a buffer REPLY of size 6 bytes to binary code and stores it in a word.

Example :

```
        JMS R2, CONVRP
        WORD RELCD
REPLY : WORD 6,0,2
        BYTE 61,62
         . = . + 4
RELCD : . = . + 2
```

Content of RELCD will be 12.

(4) Subroutine COMMFC (Common Field Check)

This subroutine checks whether a relation number RELCD exists in FDLIST. If it exists, it sets a flag CMFLFL. Also if RELCD exists in FDLIST, it compares fields of user file with fields present in relation RELCD, and their formats, and correspondingly it constructs data structures FDFLAG and CMFLDS. It also stores total number of common fields in NCOMFL. Calling instruction is JMS R7, COMMFC.

In the next few chapters, algorithms for insertion, modification and deletion operations have been described. This will complete processing phase of updating operation. Symbols used in these algorithms have the following interpretations:

NRECP = Serial No. of record currently being processed in user file.

OVERCY = Overflow cylinder number corresponding to relation RELCD, currently being updated.

ASLOCP = Pointer to starting location of primary key value in a record in sector.

SLOCR = Pointer to starting location of a record in sectors.

UPDATF = A flag which when set indicates that at least one record has been updated in the sector currently read.

OVERF = A flag which when set indicates that currently updating is going on in the overflow area.

NACTRS = No. of records in sector already tested for primary key value matching.

ACTLRC = Total No. of records in user's file.

NTIMED = No. of records updated in the sector currently read.

MAXRC = Maximum possible number of records in a sector.

COUNT 6= A counter which counts number of records in sector currently read.

BSLOCP = A pointer which points at the starting of a record in user file.

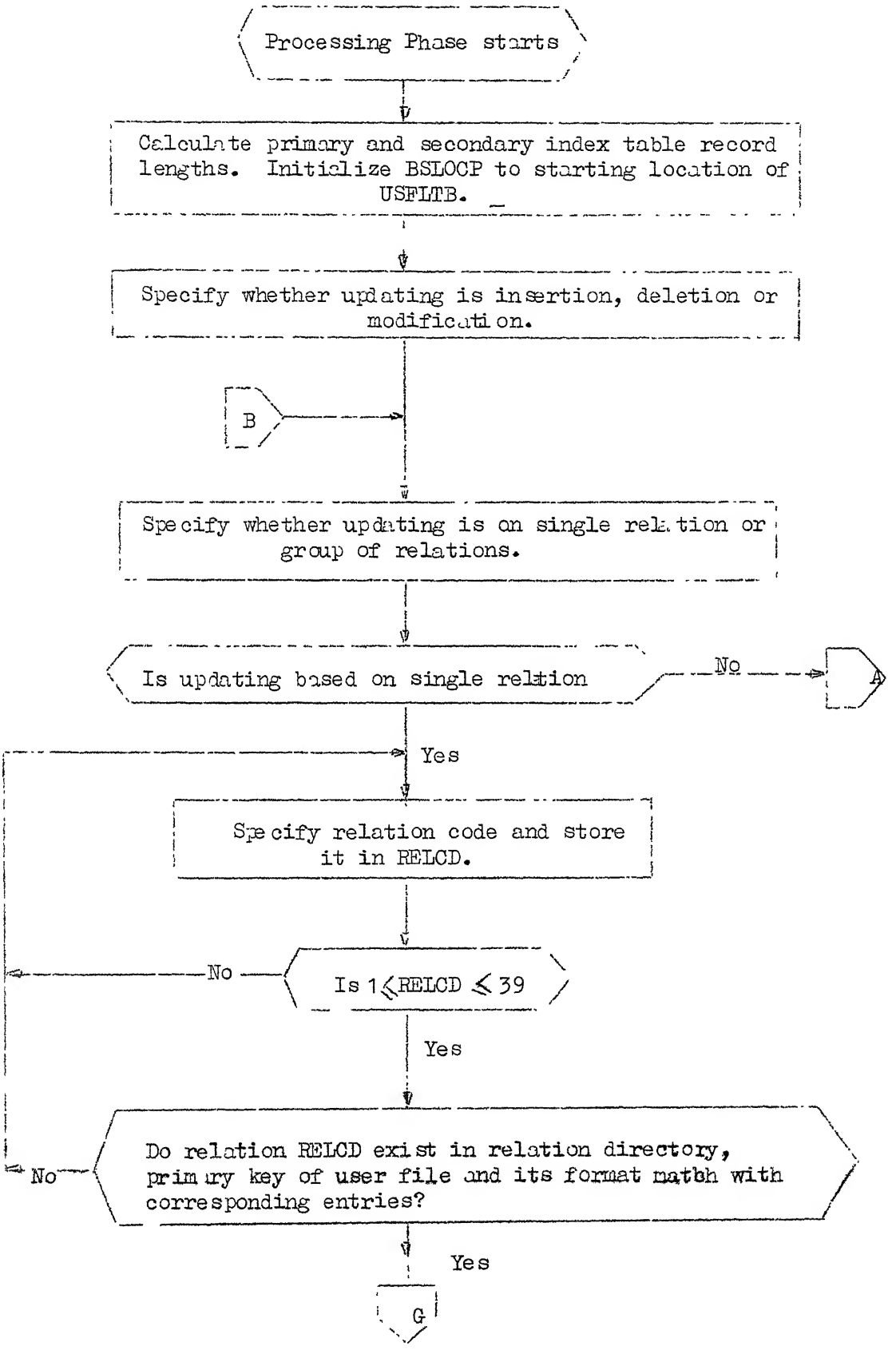
ACTLRL = Length of a record in user file.

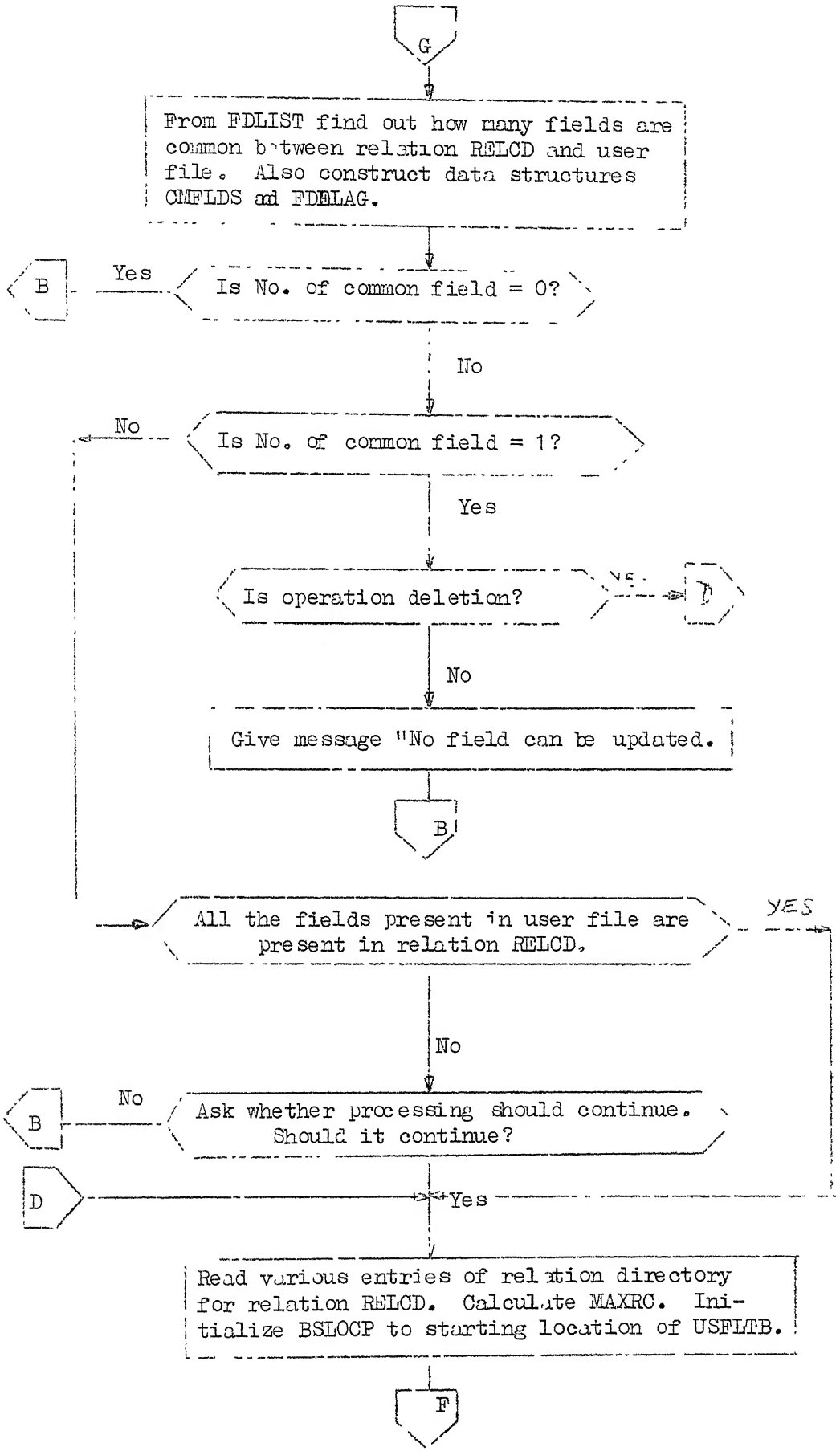
NIN = No. of records in primary index table.

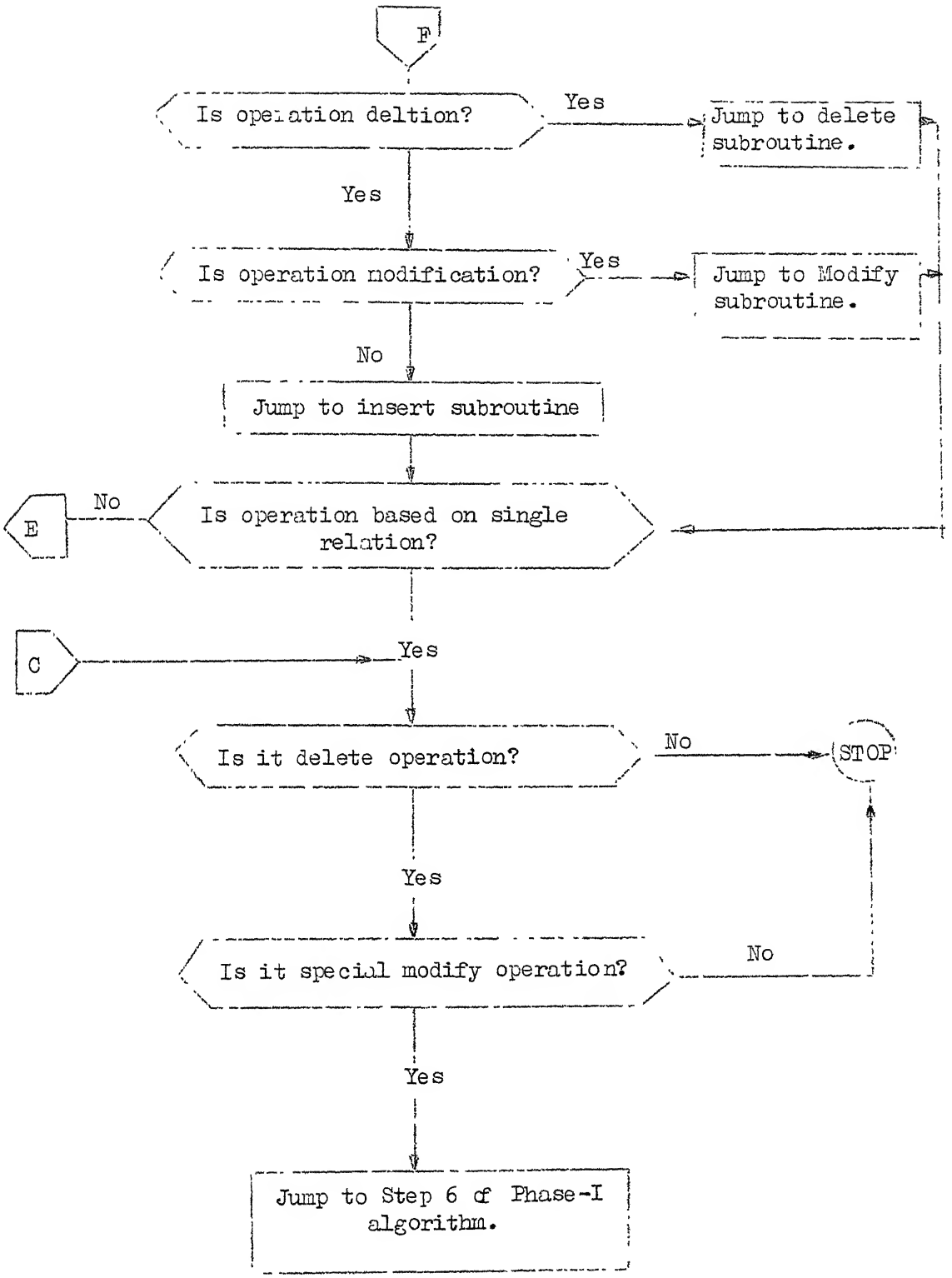
SINTL = Secondary index table record-length.

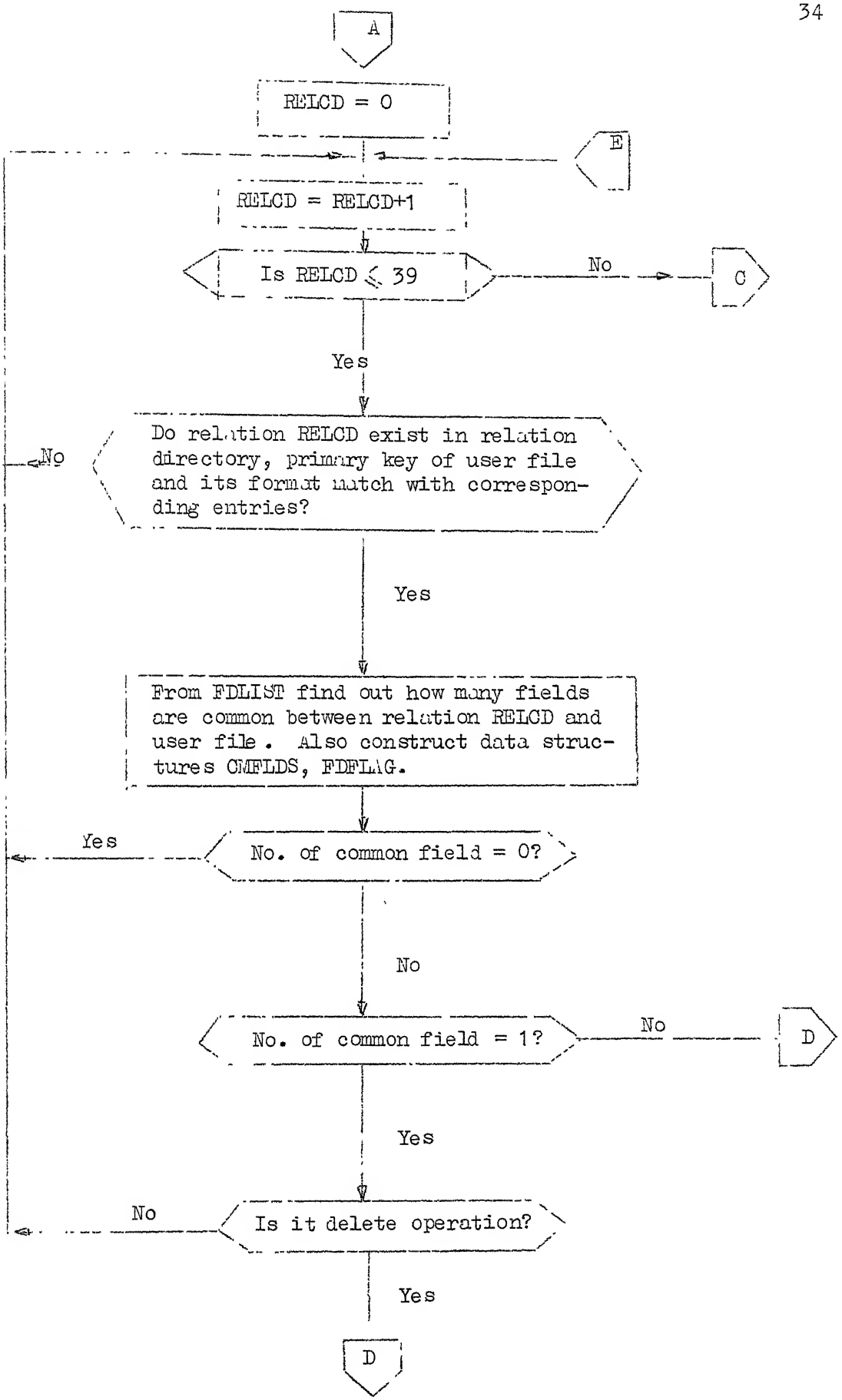
PINTL = Primary index table record-length.

Flowchart for Processing-Phase




G
From FDLIST find out how many fields are common between relation RELCD and user file. Also construct data structures CMFLDS and FDBLAG.
B
Yes
Is No. of common field = 0?
No
No
Is No. of common field = 1?
Yes
Is operation deletion?
No
D
No
Give message "No field can be updated.
B
All the fields present in user file are present in relation RELCD.
YES
No
No
B
Ask whether processing should continue. Should it continue?
D
Yes
Read various entries of relation directory for relation RELCD. Calculate MAXRC. Initialize BSLOCP to starting location of USFLTB.
F


F
Is operation deltion?
Yes
Jump to delete subroutine.
Yes
Is operation modification?
Yes
Jump to Modify subroutine.
No
Jump to insert subroutine
E
No
Is operation based on single relation?
C
Yes
Is it delete operation?
No
STOP
Yes
Is it special modify operation?
No
Yes
Jump to Step 6 of Phase-I algorithm.


A
RELCD = 0
E
RELCD = RELCD+1
Is RELCD ≤ 39
No
C
Yes
Do relation RELCD exist in relation directory, primary key of user file and its format match with corresponding entries?
No
Yes
From FDLIST find out how many fields are common between relation RELCD and user file. Also construct data structures CMFLDS, FDFLAG.
No. of common field = 0?
Yes
No
No. of common field = 1?
No
D
Yes
Is it delete operation?
No
Yes
D

5. DELETION

Deletion is an updating operation where some records from a relation or a group of relations are deleted. When records are deleted from a single relation, it is called single relation deletion and when they are deleted from a group of relations it is group relation deletion. For personnel data base, example of single relation deletion is a situation where it is observed that due to some reasons, informations stored in a relation are no longer relevant for a person and example of group relation deletion is a situation where a person leaves the organization.

In deletion operation, user file consists of only one field, the primary key of user file. If more than one field is specified all the other fields except the first one will be neglected. A sample user file is given below -

Personnel No.
IC-00050
IC-00060
IC-00065

Before giving algorithm for delete operation, a brief description of a few subroutines, written to implement this algorithm is given so that the algorithm can be presented more easily.

Subroutine READ (Read a Sector)

This subroutine stores the contents of a sector in an assigned area in core memory.

Example -

```
            JMS R4, READ
            WORD CYLNDR
            WORD SURSEC
            WORD AREA
CYLNDR:     WORD 100.
SURSEC:     WORD 1
AREA   :    . = .+400
```

Contents of Sector No. 1 of cylinder No. 100 will be stored in AREA.

Subroutine WRITE (Write on a Sector)

This subroutine stores the contents of an area of size 256 bytes in a sector.

Example -

```
            JMS R4, WRITE
            WORD CYLNDR
            WORD SURSEC
            WORD AREA
CYLNDR:     WORD 50.
SURSEC:     WORD 2
AREA   :    . = .+400
```

Contents of AREA is transferred to second sector of cylinder No. 50.

Subroutine TRSUSE (Track, Surface, Sector)

Given a primary key value as input, this subroutine searches primary and secondary index tables to find cylinder number and surface-sector address in which a record of this primary key value should exist.

If primary index table search is successful, it sets a flag MTCHPR else it is reset. If secondary index table search is successful it sets a flag MTCHSR else it is reset. If both searches are successful, it stores cylinder number in CYLNDR, total number of records in secondary index table in NRECSE, surface-sector address in SURSEC, total number of records in this sector in NRECST. Assuming first record of index tables as

Oth, record, this subroutine stores the record number of primary index table at which matching took place in COUNT0, record number of first secondary index table in COUNT1 and record number of subsequent secondary index table in COUNT3.

Example -

```
          TSR #PKYVAL, SLOCP
          JMS R3, TRSUSE
          WORD SLOCP
PKYVAL: BYTE 'I, 'C, 40,60,60,60,62,60
SLOCP : . = . + 2
```

Subroutine COMPEQ (Compare for Equality)

This subroutine compares contents of two areas for equality. If equal, it sets a flag EQFLAG else it is reset.

Example -

```
          TSR #AREA1, LOC1
          TSR #AREA2, LOC2
          JMS R2, COMPEQ
              WORD LOC1
              WORD LOC2
              WORD SIZE
AREA1:BYTE 'I,'C,40,60,60,60,62,60
AREA2:BYTE 'I,'C,40,60,60,60,62,60
SIZE : WORD 8.; size to be compared.
```

In this case EQFLAG is set.

Subroutine BLANKA (to Blank Area)

This subroutine puts blanks in a given area.

Example :

```
          TSR #BUF1, SLOC
          TSR #10., SIZE
          JMS R2, BLANKA
              WORD SLOC; starting location
              WORD SIZE
BUF1:  . = . + 100
SIZE:  . = . + 2
```

Contents of first 10. bytes of BUF1 will be blanks.

Subroutine SUSE (Surface-Sector)

Given a primary key value and cylinder number of a cylinder in which a record corresponding to this primary key value exists, this subroutine searches secondary index table. If search is successful it sets flag MTCHSR and stores furface-sector number in SURSEC. This subroutine is a part of subroutine TRSUSE.

Example -

```
          TSR # PKYVAL, SLOCP
              JMS R3, SUSE
              WORD SLOCP
PKYVAL: BYTE 'I,'C,40,60,60,60,62,60
SLOCP : . = . + 2
```

Algorithm for delete operation is as follows -

Step 1: Initialization, NRECP = 0; Find OVERCY.

Step 2: Initialize ASLOCP, SLOCR; Reset flags UPDATE and OVERF; NACTRS=0; NRECP = NRECP+1.

Step 3: Is NRECP > ACTLRC? If yes go to Step 4; else go to Step 5.

Step 4: Return to calling program.

Step 5: Print record No. NRECP and call subroutine TRSUSE.

Step 6: MTCHPR = set? If yes go to Step 9; else go to Step 7.

Step 7: Give message, "Primary Index Table Search fails".

Step 8: Take next record in user's file. Go to Step 2.

Step 9: MTCHSR = set? If yes go to Step 11; else go to Step 10.

Step 10: Give message, "Secondary Index Table search fails". Go to Step 8.

Step 11: Is NRECST = 0? If yes go to Step 12; else go to Step 13.

Step 12: Give message, "Record does not exist in sector". Go to Step 8.

Step 13: NTTMED = 0.

Step 14: Read sector SURSEC of cylinder No. CYLNDR.

Step 15: COUNT6 = 0.

Step 16: COUNT6 = COUNT6+1.

Step 17: Is NACTRS < NRECST? If yes go to Step 18; else go to Step 28.

Step 18: Is COUNT6 > MAXRC? If yes go to Step 23; else go to Step 19.

Step 19: Check whether primary key value is blank of current record in sector. If yes go to Step 22; else go to Step 20.

Step 20: NACTRS = NACTRS+1. Compare primary key values of this record and user file record. Are they equal? If yes go to Step 21; else go to Step 22.

Step 21: Delete current record in sector. Set UPDATF flag. NTIMED = NITEMD + 1.

Step 22: Consider next record in sector. Go to Step 16.

Step 23: Is flag UPDATF = set? If yes go to step 24; else go to Step 25.

Step 24: Write back sector.

Step 25: Is OVERCY = 0? If yes go to Step 26; else go to Step 27.

Step 26: Give message, "Secondary index table inconsistent." Go to Step 28.

Step 27: From last two bytes of current sectors read pointer to overflow sector. Set OVERF flag. Initialize ASLOCP, SLOCR. Read overflow sector. Go to Step 15.

Step 28: Is UPDATF = set? If yes, go to Step 29; else go to Step 12.

Step 29: Write back sector. Update secondary index table.

Step 30: Compare primary key value of user file record with highest primary key entry in current record of secondary index table. Are they equal? If yes go to Step 33; else go to step 31.

Step 31: Write back secondary index-table.

Step 32: Give message, "Record Processed". Go to Step 8.

Step 33: COUNT3= COUNT3+1
Is the current record of secondary index table last record? If yes go to Step 38; else go to Step 34.

Step 34: Is primary key value of record No. COUNT3 in secondary index table $\geq$ primary key value of user file-record? If yes go to Step 35; else go to Step 31.

Step 35: Read SURSEC and NRECST from current record of secondary index table.

Step 36: Is NRECST = 0? If yes go to Step 30; else go to Step 37.

Step 37: Initialize ASLOCP, SLOCR; NACTRS = 0, NTIMED = 0; go to Step 14.

Step 38: Write back secondary index table. COUNT1 = COUNT+1. Read secondary index table from first sector. Is highest primary key entry in record No. COUNT1 blank? If yes go to Step 39, else go to Step 40.

Step 39: Read NRECSE and SURSEC from current record in secondary index table. Read secondary index table from sector SURSEC. COUNT3 = 0. Go to Step 34.

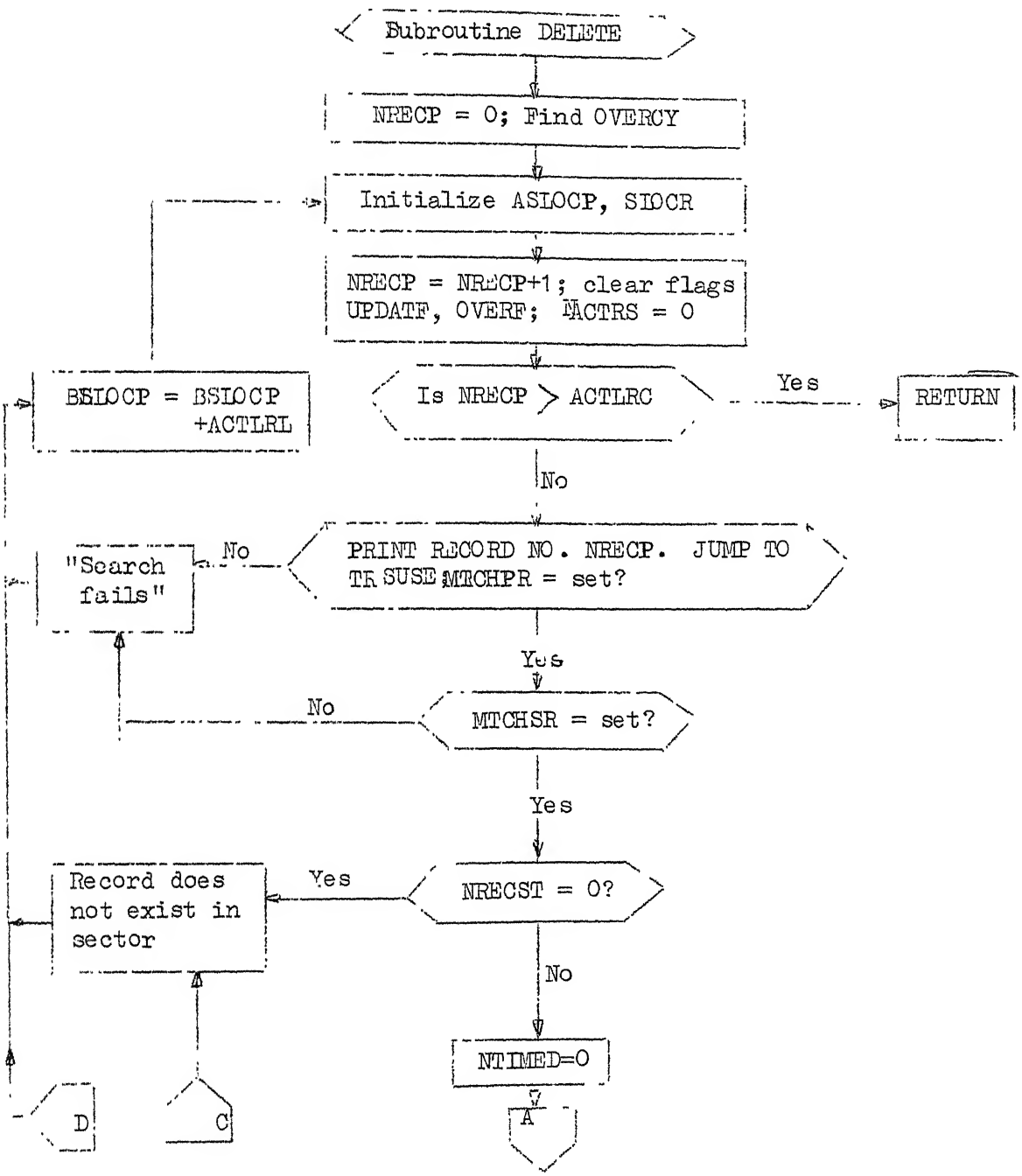
Step 40: COUNTO = COUNTO+1
Is COUNTO = NIN? If yes, go to Step 32; else go to Step 41.

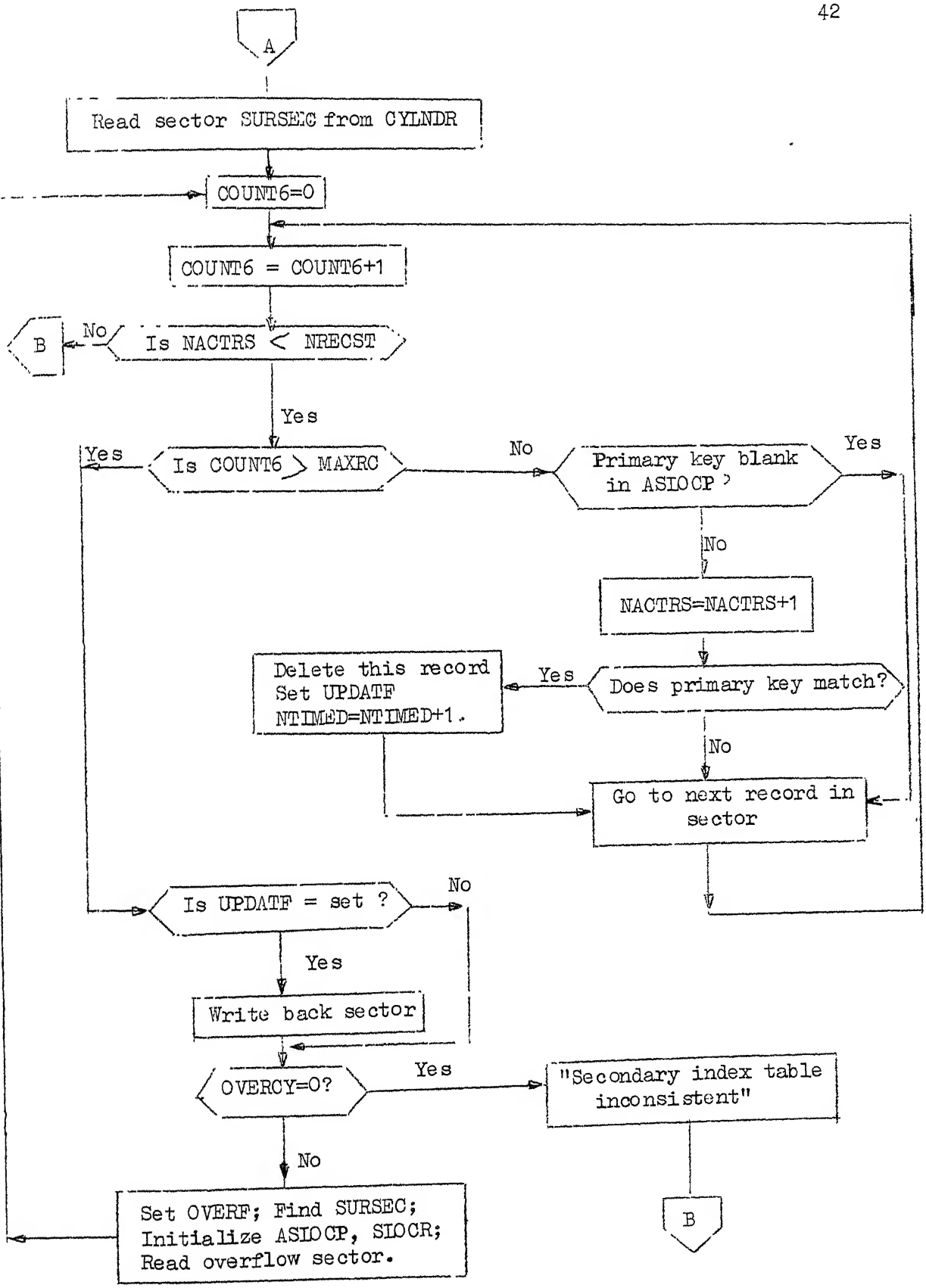
Step 41: Is highest primary key values entry in current record of primary index table $\geq$ primary key value of user file record? If yes go to Step 42; else go to Step 32.

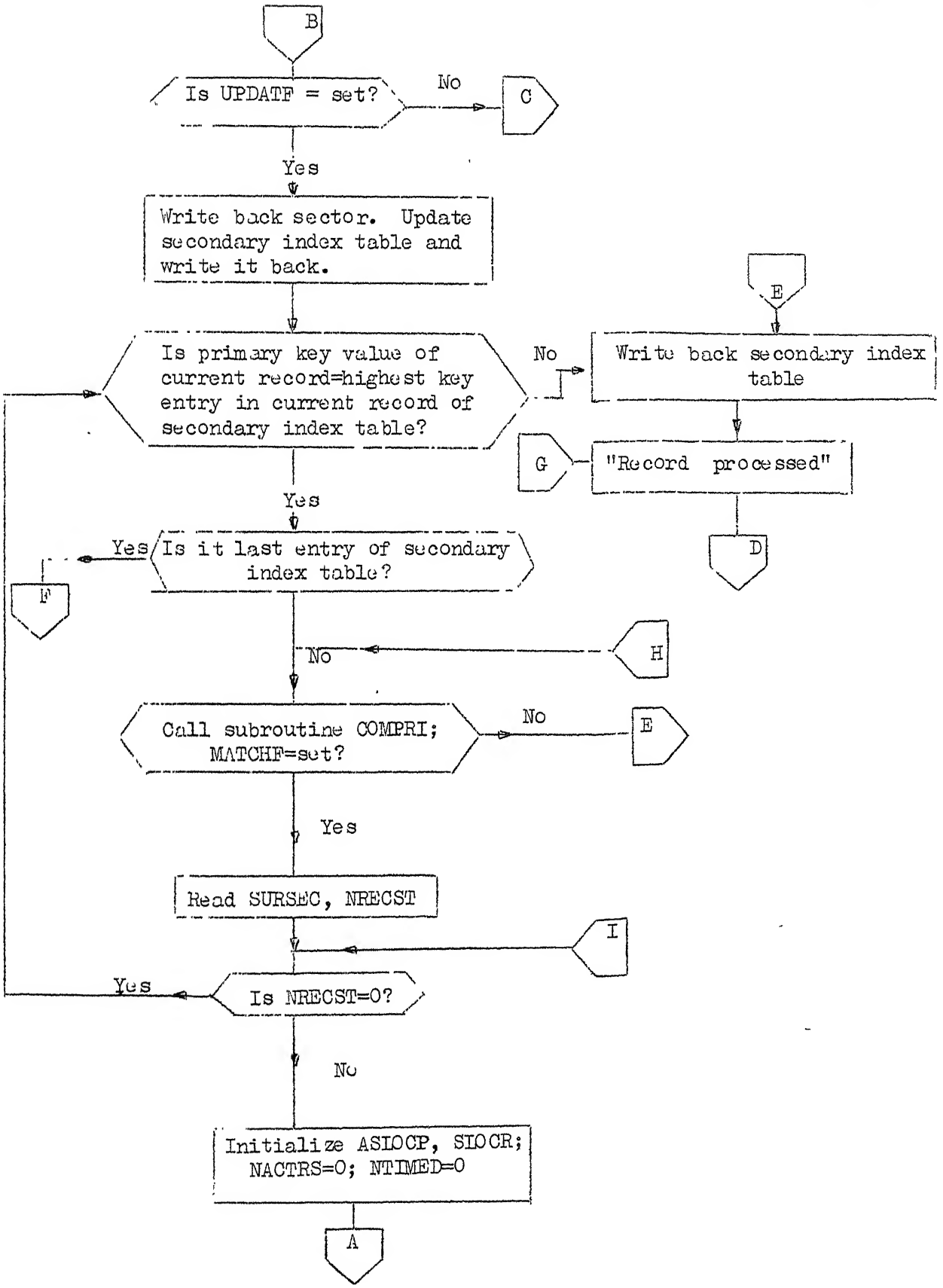
Step 42: Read CYLNDR, NRECSE from current record of primary index table. Call subroutine SUSE to find SURSEC. MTCHSR = set? If yes, go to Step 36; else go to Step 32.

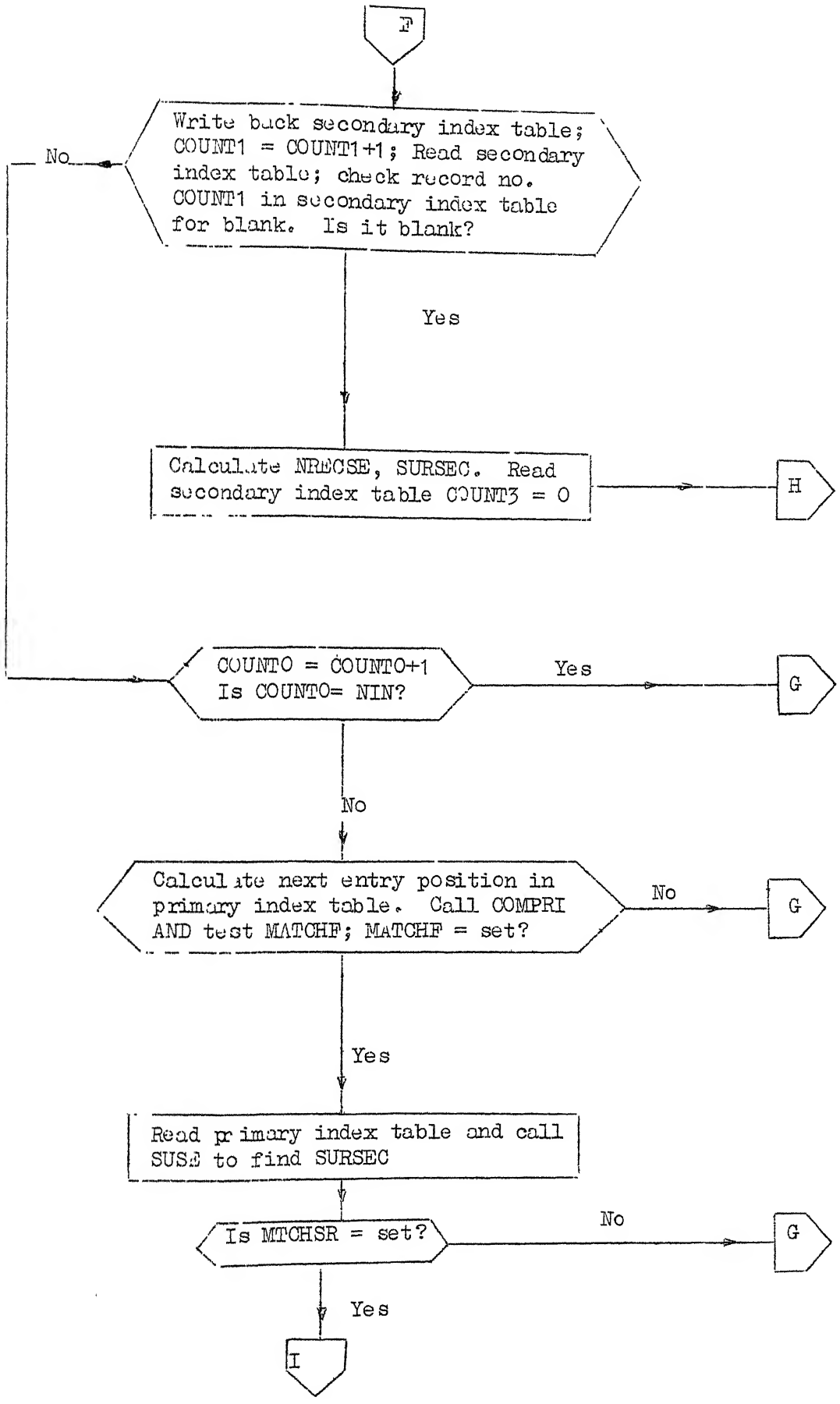
End of Algorithm.

Flowchart for DELETE




A
Read sector SURSEC from CYLNDR
COUNT6=0
COUNT6 = COUNT6+1
B
No
Is NACTRS < NRECST
Yes
Yes
Is COUNT6 > MAXRC
No
Primary key blank in ASIOCP ?
Yes
No
NACTRS=NACTRS+1
Delete this record
Set UPDATF
NTIMED=NTIMED+1.
Yes
Does primary key match?
No
Go to next record in sector
Is UPDATF = set ?
No
Yes
Write back sector
OVERCY=0?
Yes
"Secondary index table inconsistent"
No
Set OVERF; Find SURSEC;
Initialize ASIOCP, SIOCR;
Read overflow sector.
B


B
Is UPDATF = set?
No
C
Yes
Write back sector. Update secondary index table and write it back.
E
Is primary key value of current record=highest key entry in current record of secondary index table?
No
Write back secondary index table
G
"Record processed"
D
Yes
Yes
Is it last entry of secondary index table?
F
No
H
Call subroutine COMPRI; MATCHF=set?
No
E
Yes
Read SURSEC, NRECST
I
Yes
Is NRECST=0?
No
Initialize ASIOCP, SIOCR; NACTRS=0; NTIMED=0
A


F
Write back secondary index table; COUNT1 = COUNT1+1; Read secondary index table; check record no. COUNT1 in secondary index table for blank. Is it blank?
No
Yes
Calculate NRECSE, SURSEC. Read secondary index table COUNT3 = 0
H
COUNTO = COUNTO+1 Is COUNTO= NIN?
Yes
G
No
Calculate next entry position in primary index table. Call COMPRI AND test MATCHF; MATCHF = set?
No
G
Yes
Read primary index table and call SUSE to find SURSEC
Is MTCHSR = set?
No
G
Yes
I

6. MODIFICATION

Modification is an updating operation where values of different fields of some records are modified. When this operation is performed on a single relation it is called single relation modification and when it is performed on all the relations it is called group relation modification. For personnel data base, example of modification operation is a situation when some persons get promotion and their ranks and salary are to be updated. If all these fields except the primary key occur only in one relation it is single relation modification and if these fields occur in more than one relation or different relations it is group relation modification.

In modification operation user file consists of primary key and the fields to be modified. A sample user file is as follows -

Personnel No.	Rank	Salary	Qualif.
IC-00050	CAPT.	1600	*
IC-00056	MAJOR	1400	MA.A.
IC-00110	*	*	M.Tech.

Symbol * (star) as value of a field indicates that value of this field remains unmodified for the corresponding primary key value.

Algorithm for modification is similar to algorithm for deletion except a minor change that instead of deleting a complete record, values of different fields are modified. Subroutines used in modification operation are already explained in previous chapters except the following -

(1) Subroutine TRANS (Transfer)

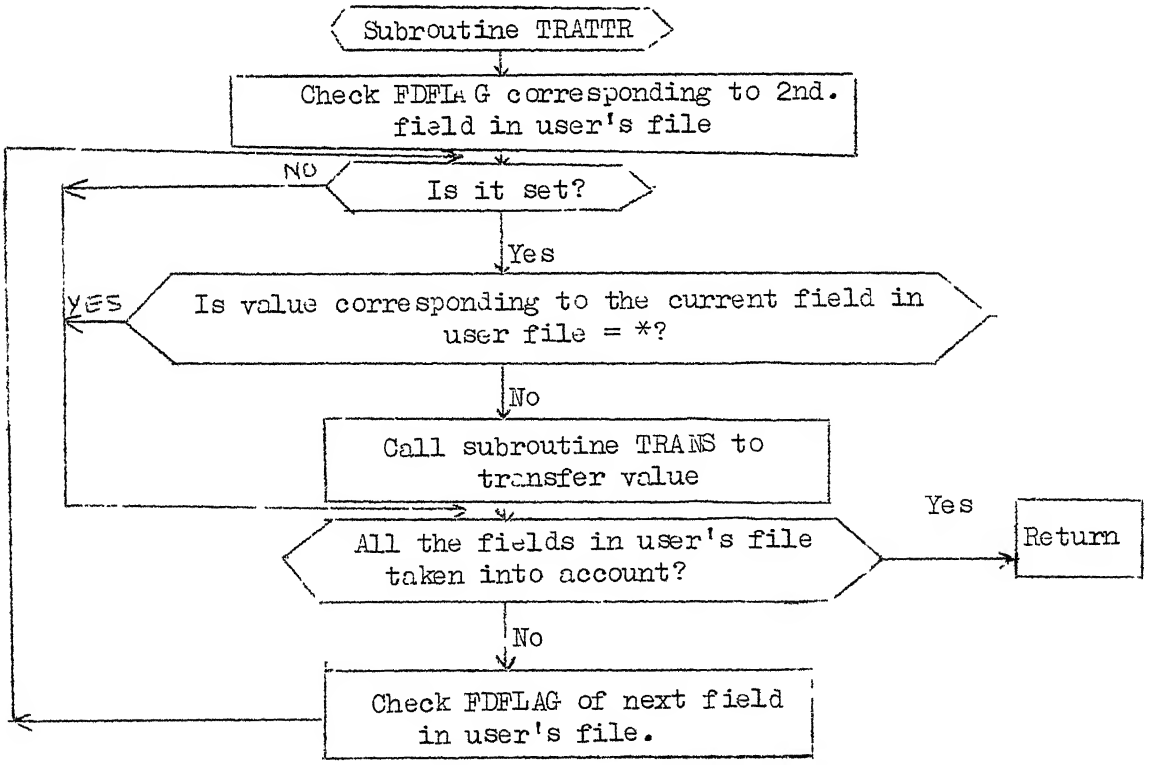
This subroutine transfers the contents of an area of a specified size with specified starting location to another area whose starting location is specified. Example -

```
        TSR #SOURCE, LOCS
        TSR #DESTN, LOCD
            JMS R5, TRANS
            WORD LOCS
            WORD LOCD
            WORD SIZE
SOURCE  : BYTE 1,5,9., 10., 12, 4
DESTN   : . = . + 100
 SIZE   : WORD 4
```

Contents of first 4 bytes of DESTN will be 1,5,9., 10. respectively.

(2) Subroutine TRATTR (Transfer attributes)

This subroutine modifies a complete record. Flowchart for this subroutine is given below.



Calling Instruction of this Subroutine is JMS R2, TRATTR.

Algorithm for modification operation is as follows -

Step 1: Intialize NRECP = 0. Find overflow cylinder number corresponding to relation RELCD and store it in OVERCY.

Step 2: Initialize ASLOCP, SLOCR. Increment NRECP by 1. Reset flags UPDATF, OVERF. Initialize NACTRS = 0.

Step 3: Is NRECP > ACTLRS? If yes go to Step 4; else go to Step 5.

Step 4: Return to calling program.

Step 5: Print record number NRECP and call subroutine TRSUSL.

Step 6: MTCHPR = set? If yes go to Step 9; else go to Step 7.

Step 7: Give message "Primary index table search fails."

Step 8: Take next record in user's file. Go to Step 2.

Step 9: MTCHSR = set? If yes go to Step 11; else go to Step 10.

Step10: Give message "Secondary index table search fails". Go to Step 8.

Step 11: Is NRECST = 0? If yes go to Step 12; else go to Step 13.

Step 12: Give message "Record does not exist in sector". Go to Step 8.

Step 13: Read sector SURSEC of cylinder number CYLNDR.

Step 14: COUNT 6 = 0.

Step 15: COUNT 6 = COUNT 6+1.

Step 16: Is NACTRS < NRECST? If yes go to Step 17; else go to Step 27.

Step 17: Is COUNT6 > MAXRC? If yes go to Step 22; else go to Step 18.

Step 18: Check whether primary key value is blank in current record of sector. If yes go to Step 21; else go to Step 19.

Step 19: NACTRS = NACTRS + 1.
Compare primary key value of this record and that of current record in user file. Are they equal? If yes go to Step 20; else go to Step 21.

Step 20: Call subroutine TRATTR to modify this record and set flag UPDATF in case there is any modification.

Step 21: Consider next record in sector. Go to Step 15.

Step 22: Is flag UPDATF = set? If yes go to Step 23; else go to Step 24.

Step 23: Write back sector.

Step 24: Is OVERCY = 0? If yes go to Step 25; else go to Step 26.

Step 25: Give message "Secondary index table inconsistent". Go to Step 27.

Step 26: Set flag OVERF. From last two bytes of current sector read pointer to overflow sector. Initialize ASLOCP, SLOCR. Read overflow sector. Go to Step 14.

Step 27: Is UPDATF = set? If yes go to Step 28; else go to Step 12.

Step 28: Write back sector.

Step 29: Compare primary key value of current record in user file with highest key entry in current record of secondary index table. Are they equal? If yes go to Step 31; else go to Step 30.

Step 30: Give message, "Record processed". Go to Step 8.

Step 31: COUNT3 = COUNT3+1.
Is the current record of secondary index table last record? If yes go to Step 36; else go to Step 32.

Step 32: Is primary key of current record in user file highest key entry in record No. COUNT3 of secondary index table? If yes go to Step 33; else go to Step 30.

Step 33: Read SURSEC, NRECST from secondary index table.

Step 34: Is NRECST = 0? If yes go to Step 29; else go to Step 35.

Step 35: Initialize ASLOCP, SLOCR. Initialize, NACTRS = 0. Go to Step 13.

Step 36: COUNT1 = COUNT1+1.
Read secondary index table from first sector of cylinder number CYLNDR. Is highest key value entry in record number COUNT1 blank? If yes go to Step 37; else go to Step 38.

Step 37: Read NRECSE and SURSEC from current record in secondary index table. Read secondary index table from sector SURSEC. Initialize COUNT3 = 0. Go to Step 32.

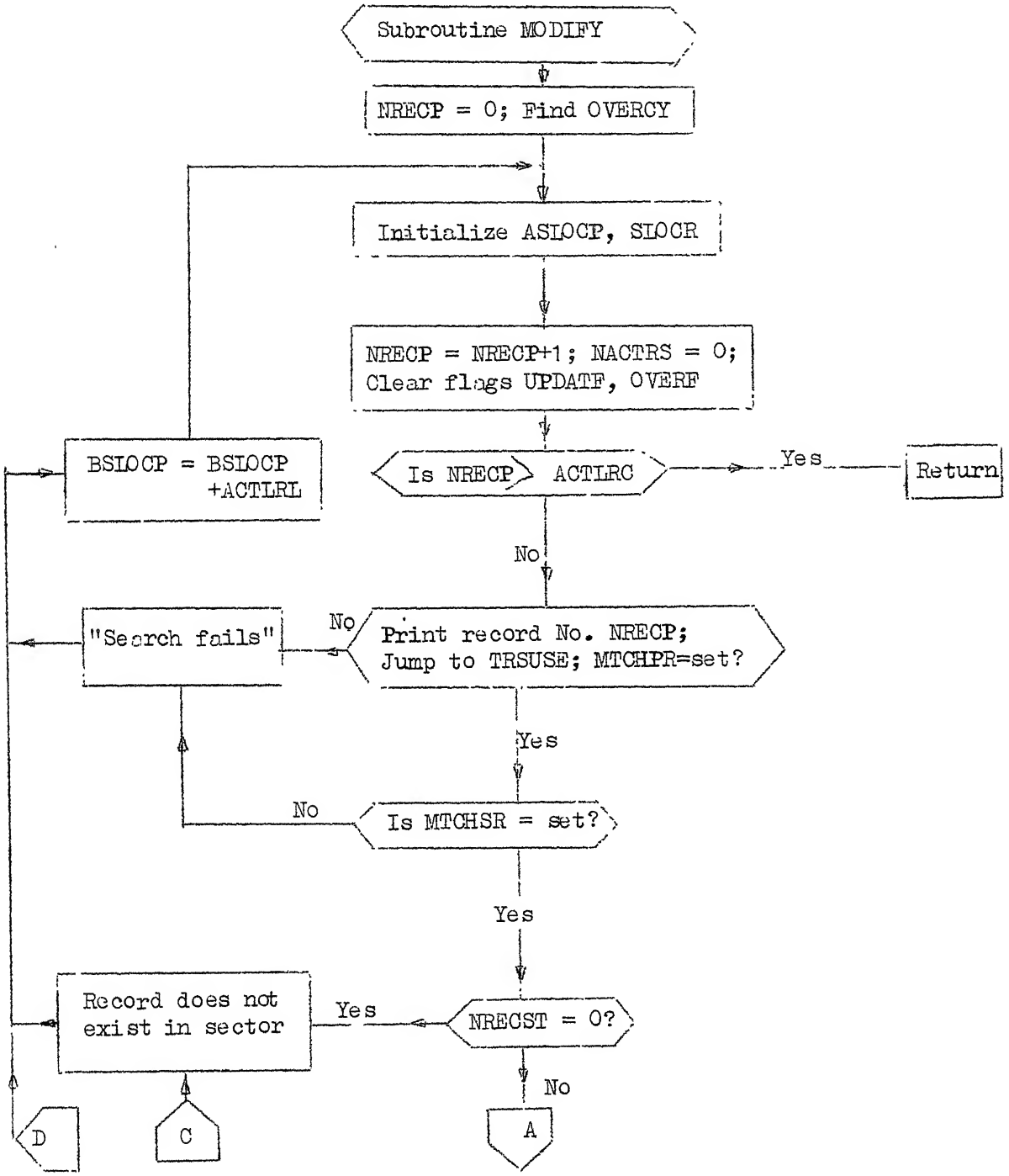
Step 38: COUNTO = COUNTO+1.
Is COUNTO = NIN? If yes go to Step 30; else go to Step 39.

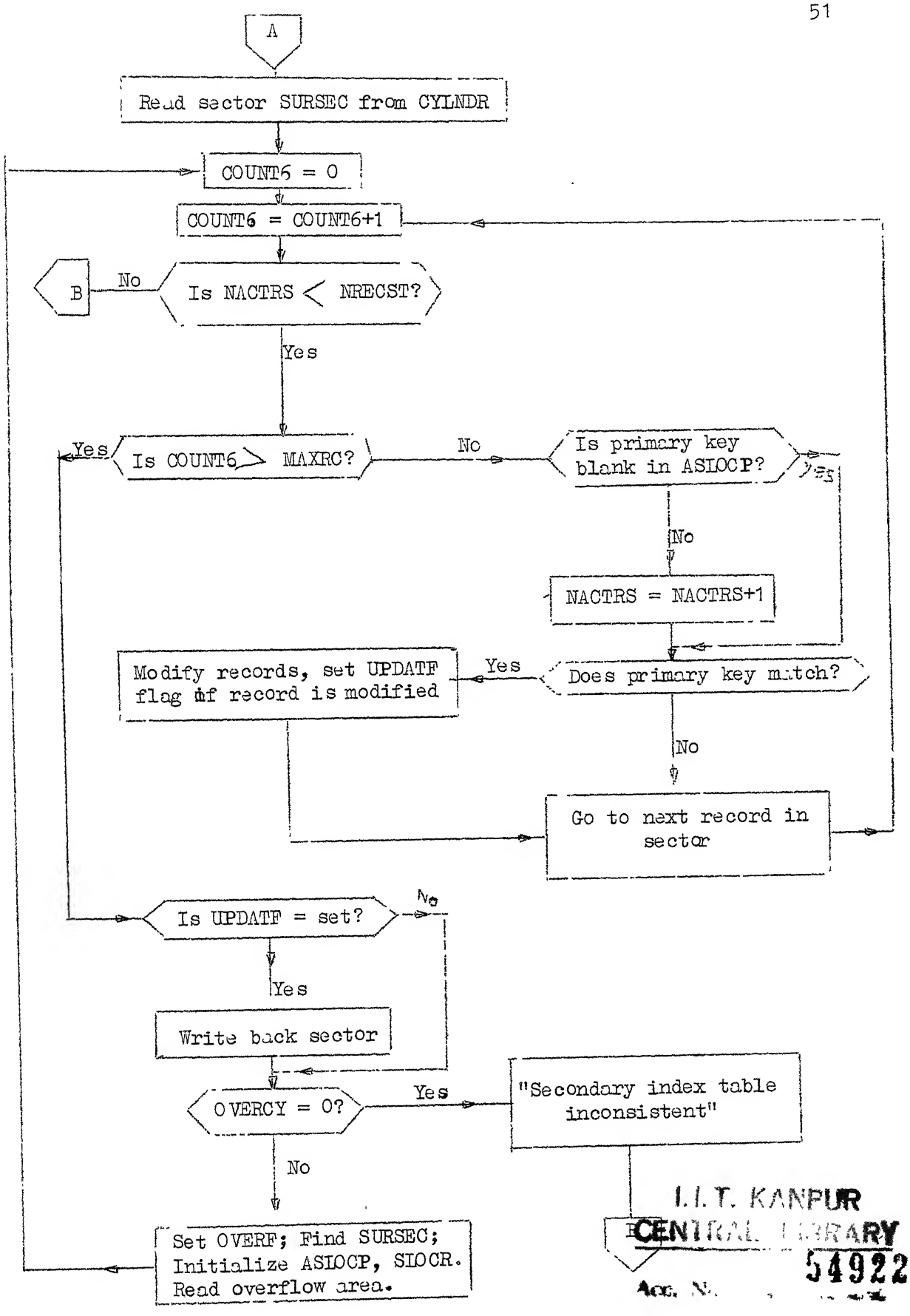
Step 39: Is highest key entry in current record of primary index table $\geq$ primary key value of current record in user file? If yes go to Step 40; else go to Step 30.

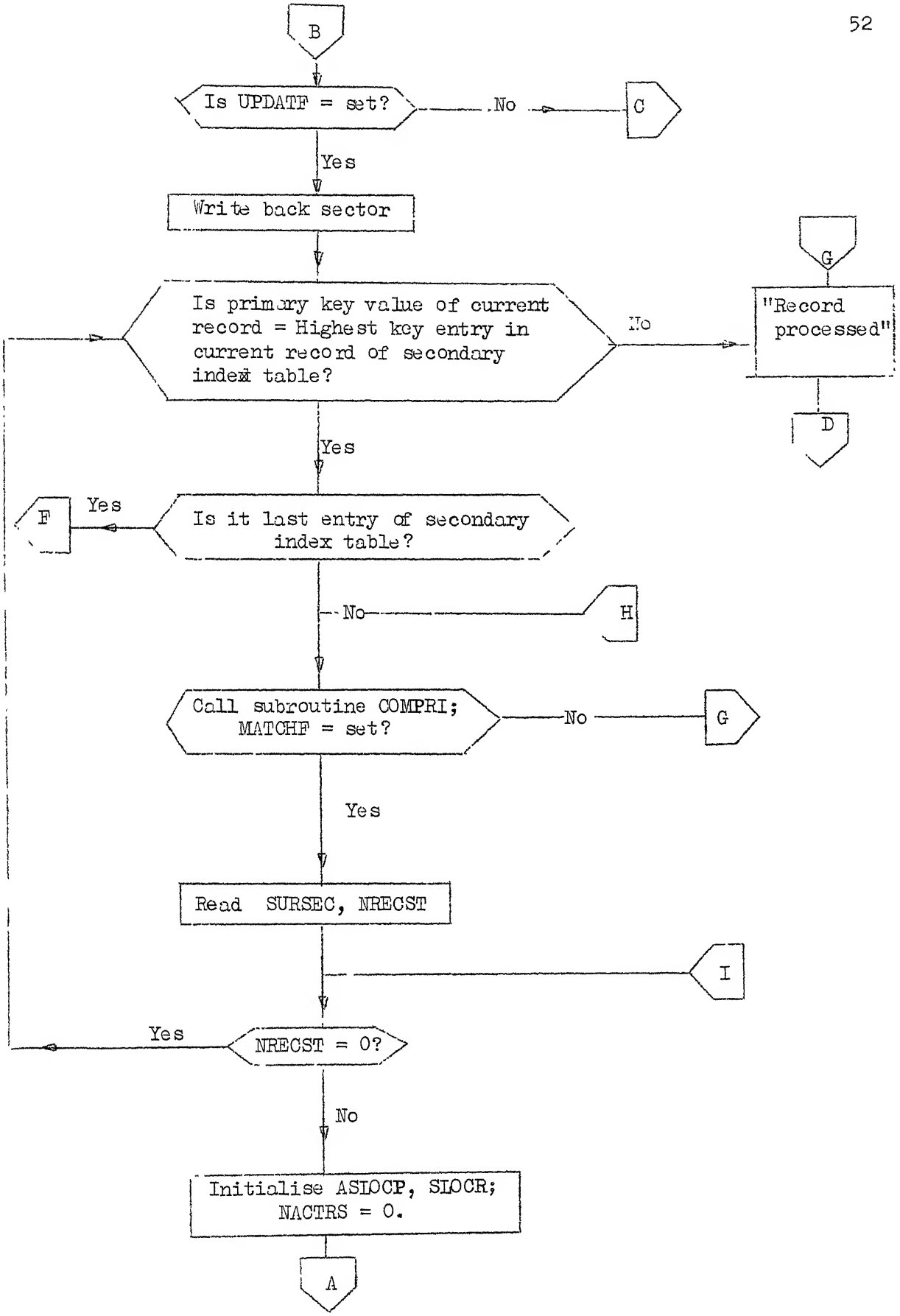
Step 40: Read CYLNDR, NRECSE from primary index table. Call subroutine SUSE to find SURSEC. Is flag MTCHSR = set? If yes go to Step 34; else go to Step 30.

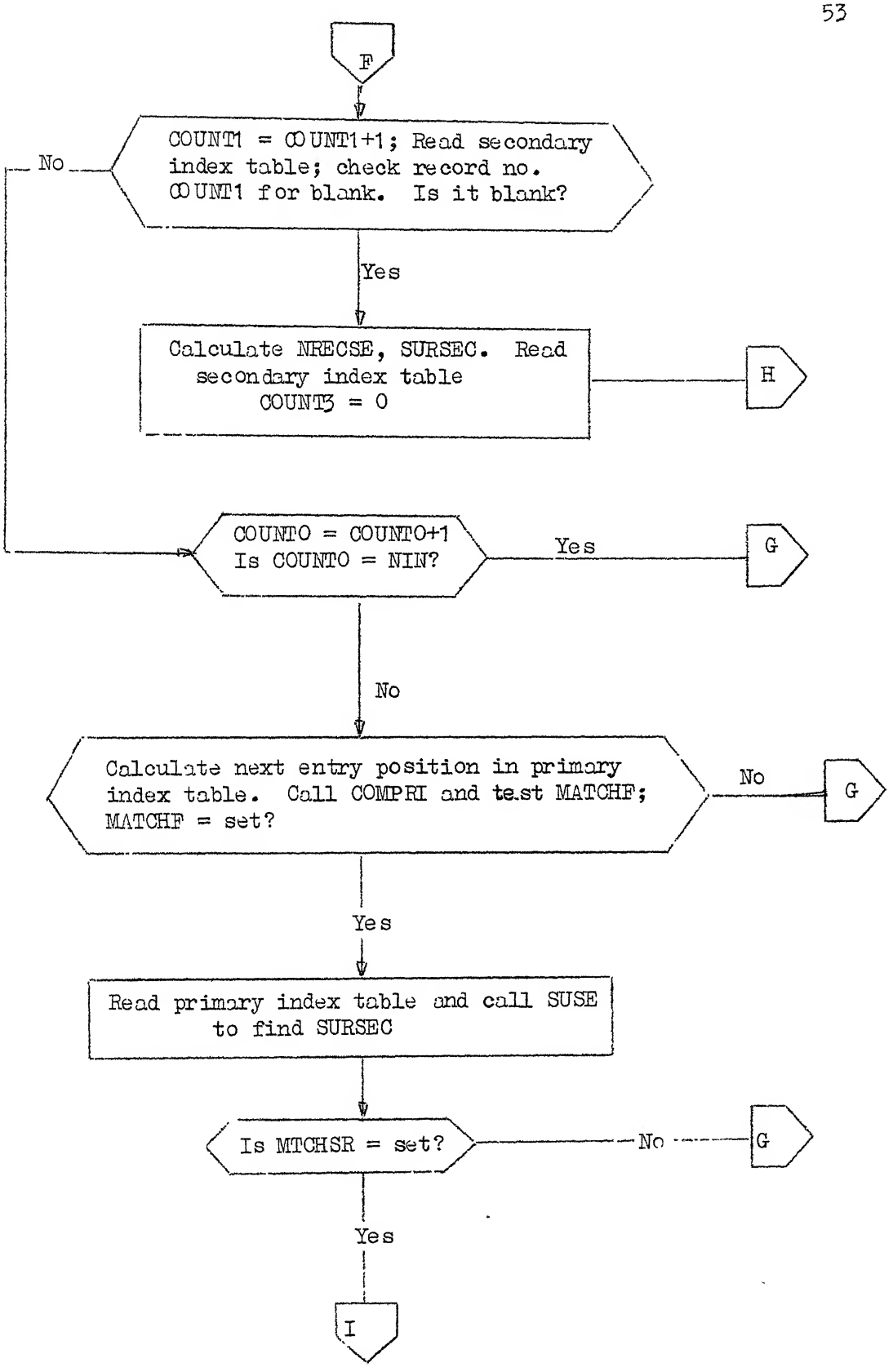
End of Algorithm.

Flowchart for Modification






B
Is UPDATF = set?
No
C
Yes
Write back sector
G
Is primary key value of current record = Highest key entry in current record of secondary index table?
No
"Record processed"
D
Yes
F
Yes
Is it last entry of secondary index table?
No
H
Call subroutine COMPRI; MATCHF = set?
No
G
Yes
Read SURSEC, NRECST
I
Yes
NRECST = 0?
No
Initialise ASLOCP, SLOCR; NACTRS = 0.
A


F
COUNT1 = COUNT1+1; Read secondary index table; check record no. COUNT1 for blank. Is it blank?
No
Yes
Calculate NRECSE, SURSEC. Read secondary index table
COUNT3 = 0
H
COUNT0 = COUNT0+1
Is COUNT0 = NIN?
Yes
G
No
Calculate next entry position in primary index table. Call COMPRI and test MATCHF; MATCHF = set?
No
G
Yes
Read primary index table and call SUSE to find SURSEC
Is MTCHSR = set?
No
G
Yes
I

7. INSERTION

Insertion is an updating operation where some new records are inserted in data base. When records are inserted in a single relation, it is single relation insertion and when they are inserted in more than one relation, it is group relation insertion. A sa user file for insertion operation is as follows -

Personnel No.	Rank	Salary	Qualification
IC-00050	CAPT.	1600	*
IC-00056	MAJOR	1400	M.A.
IC-00110	*	*	M.Tech.

If all the fields specified in user file except the primary key occur only in one relation it is single relation insertion and if they occur in different relations or more than one relation it is group relation insertion. Symbol * (Star) as value of a field indicates that value of this field is left blank for the corresponding primary key while inserting this record. Different subroutines used in insertion operation are explained in previous chapters except the followings:

(1) Subroutine ALLOCT (Allocate a Cylinder)

This subroutine finds out cylinder number of the first available empty cylinder in disk starting from cylinder number 0 and stores it in OVERCY. This cylinder is used as overflow cylinder. Calling instruction is JMS R4, ALLOCT.

(2) Subroutine NXTSEC (Next Sector)

This subroutine finds the sector number of a sector next to sector number SURSEC and stores it in NEWSS. Calling instruction JMS R4, NXTSEC.

(3) Subroutine PUT

This subroutine writes back primary index table on disk. Calling instruction is JMS R , PUT.

Algorithm for insertion operation is given below:

Step 1: Initialize NRECP = 0. Find overflow cylinder number corresponding to relation RELCD and store it in OVERCY.

Step 2: Initialize ASLOCP, SLOCR. Increment NRECP by 1. Clear flags FOUNDF, OVERF, PINDXF.

Step 3: Is NRECP > ACTLRS? If yes go to Step 4; else go to Step 7.

Step 4: Is PINDXF flagset? If yes go to Step 5; else go to Step 6.

Step 5: Write back primary index table on disk.

Step 6: Return to calling program.

Step 7: Print Record No. NRECP.

Step 8: Call subroutine TRSUSE to find cylinder and sector no.

Step 9: Is MTCHPR = set? If yes go to Step 11; else go to Step 10.

Step 10: Set PINDX flag. Transfer primary key value of current record in user file to the last record in primary index table. Go to Step 8.

Step 11: Is MTCHSR = set? If yes go to Step 13; else go to Step 12.

Step 12: Transfer primary key value of current record in user file to last record in current secondary index table. Go to Step 8.

Step 13: CRNTCY ← CYLNDR.

Step 14: Read sector no. SURSEC from cylinder no. CYLNDR.

Step 15: COUNT6 = 0.

Step 16: COUNT6 = COUNT6+1.

Step 17: Is COUNT6 >MAXRC? If yes go to Step 26; else go to Step 18.

Step 18: Is primary key value in current record of sector blank? If yes, go to Step 22; else go to Step 19.

Step 19: Is primary key value of current record in sector = primary key value of current record in user file? If yes go to Step 20; else go to Step 21.

Step 20: Set Flag FOUNDF.

Step 21: Update ASLOCP, SLOCR. Go to Step 16.

Step 22: Transfer current record in user file to current record in sector. Write back sector on disk.

Step 23: Update secondary index table and write it back on disk.

Step 24: Give message, "Record processed."

Step 25: Take next record in user file. Go to Step 2.

Step 26: Initialise ASLOCP, SLOCR. Is OVERCY = 0? If yes, go to Step 29; else go to Step 27.

Step 27: Read last 2 bytes of sector and store it in NEWSS. Is NEWSS = 20040? If yes go to Step 29; else go to Step 28.

Step 28: CRNTCY ← CYLNDR
SURSEC ← NEWSS.
Set flag OVLRF. Read sector no. SURSEC from cylinder CRNTCY.
Go to Step 15.

Step 29: Is flag FOUNDF = set? If yes go to Step 42; else go to Step 30.

Step 30: Set flag OVERF. Is OVERCY = 0? If yes go to Step 37; else go to Step 31.

Step 31: Read first sector from cylinder OVERCY. Read first 2 bytes of this sector into NEWSS. Is NEWSS = last sector of a cylinder? If yes, go to Step 32; else go to Step 33.

Step 32: Give message "No empty sector available in overflow cylinder". Go to Step 25.

Step 33: Find out next available sector in overflow cylinder and store it in NEWSS.

Step 34: Store NEWSS in first two bytes of first sector of overflow cylinder.

Step 35: Store NEWSS to last two bytes of sector SURSEC of cylinder CRNTCY.

Step 36: SURSEC ← NEWSS.
Fill up sector area with blanks. Go to Step 22.

Step 37: Find out the first available empty cylinder in disk. Is it available? If yes go to Step 39; else go to Step 38.

Step 38: Give message, "No empty cylinder availble in disk". Go to Step 25.

Step 39: Store cylinder no. of newly available cylinder in OVERCY. Store OVERCY in relation directory. Store 2 in last 2 bytes of sector SURSEC of cylinder no. CYLNDR.

Step 40: SURSEC ← 1
Fill up sector area by blanks. Store 2 in first two bytes of first sector of cylinder OVERCY.

Step 41: Fill up sector area by blanks. SURSEC ← 2. Go to Step 22.

Step 42: Compare primary key value of current record in user file with highest key entry in current record of secondary index table. Are they equal? If yes go to Step 43; else go to Step 30.

Step 43: COUNT3 = COUNT3+1.
Is the current record of secondary index table last record? If yes, go to Step 46; else go to Step 44.

Step 44: Ishighest key entry in current record of primary index table primary key value of currant record in user file? If yes go to Step 45; else go to Step 30.

Step 45: Read SURSEC from secondary index table. Go to Step 14.

Step 46: COUNT1 = COUNT1+1.
Read secondary index table from first sector of cylinder no. CYLNDR. Is highest key value in record no. COUNT1 blank? If yes, go to Step 47; else go to Step 48.

Step 47: Read SURSEC from secondary index table. COUNT3 = 0. Go to Step 44.

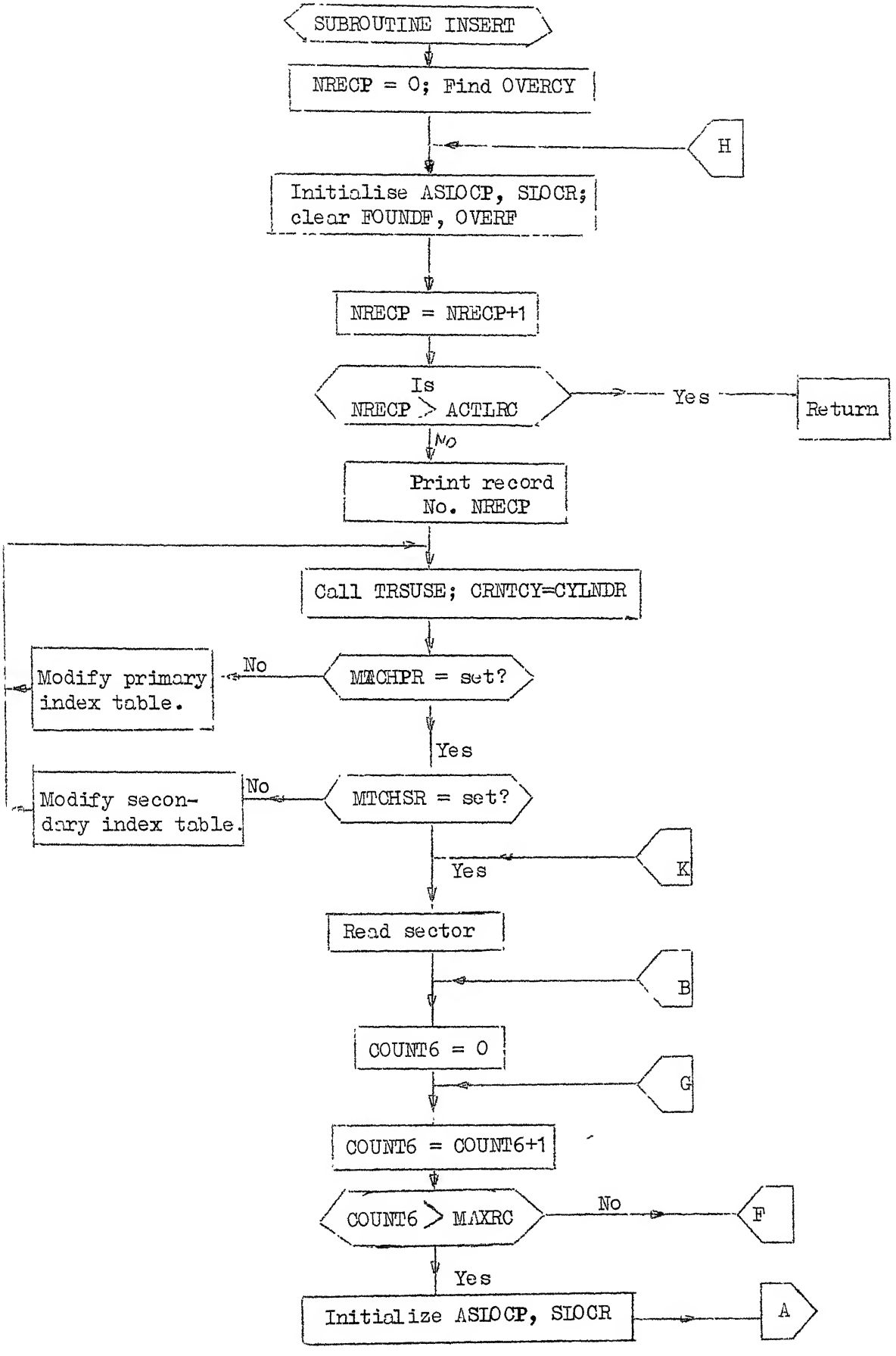
Step 48: COUNT 0 = COUNTO + 1.
Is COUNTO = NIN? If yes go to Step 30; else go to Step 49.

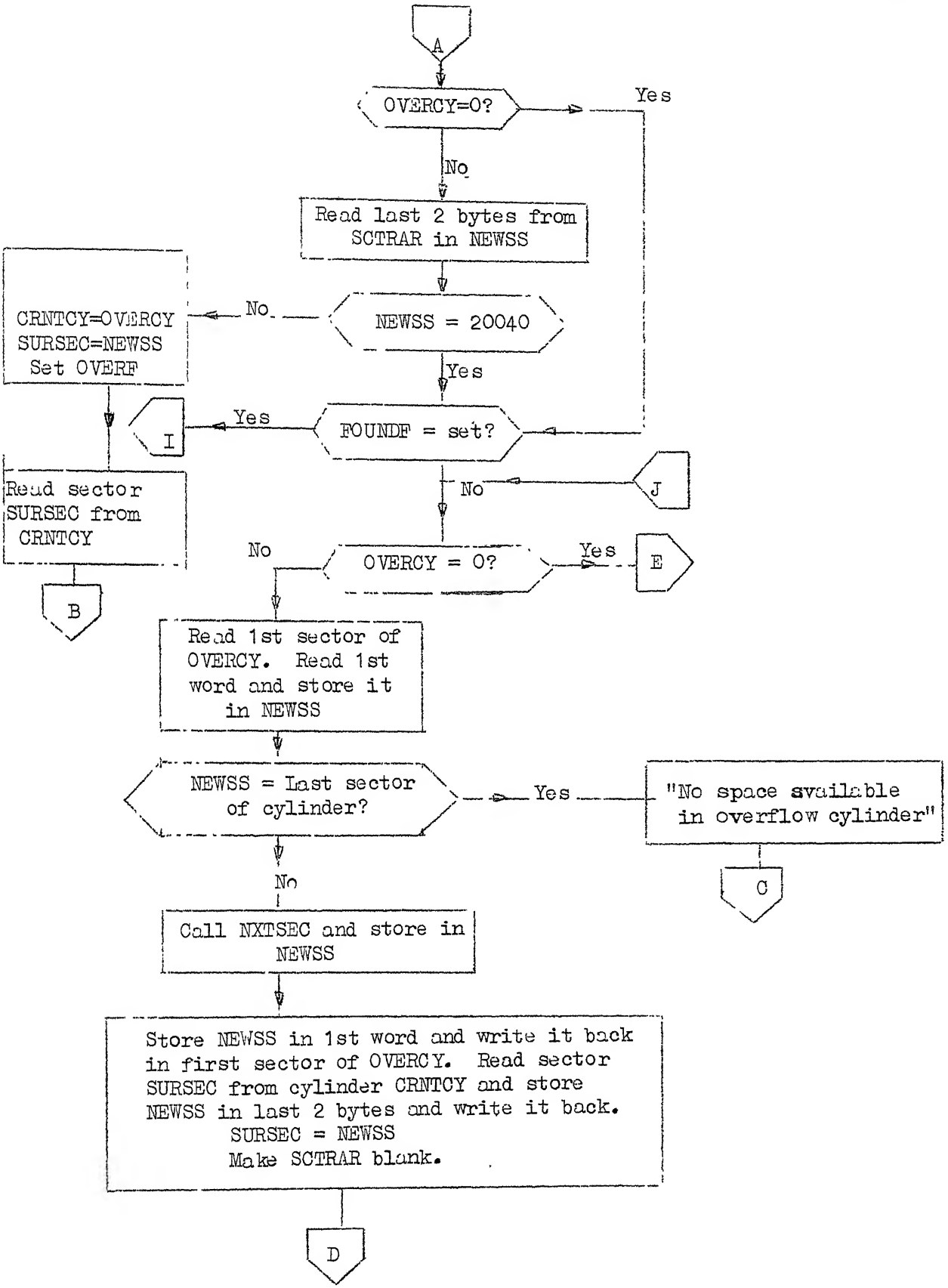
Step 49: Is highest primary key entry in current record of primary index table $\geq$ primary key value of current record in user file? If yes, go to Step 50, else go to Step 30.

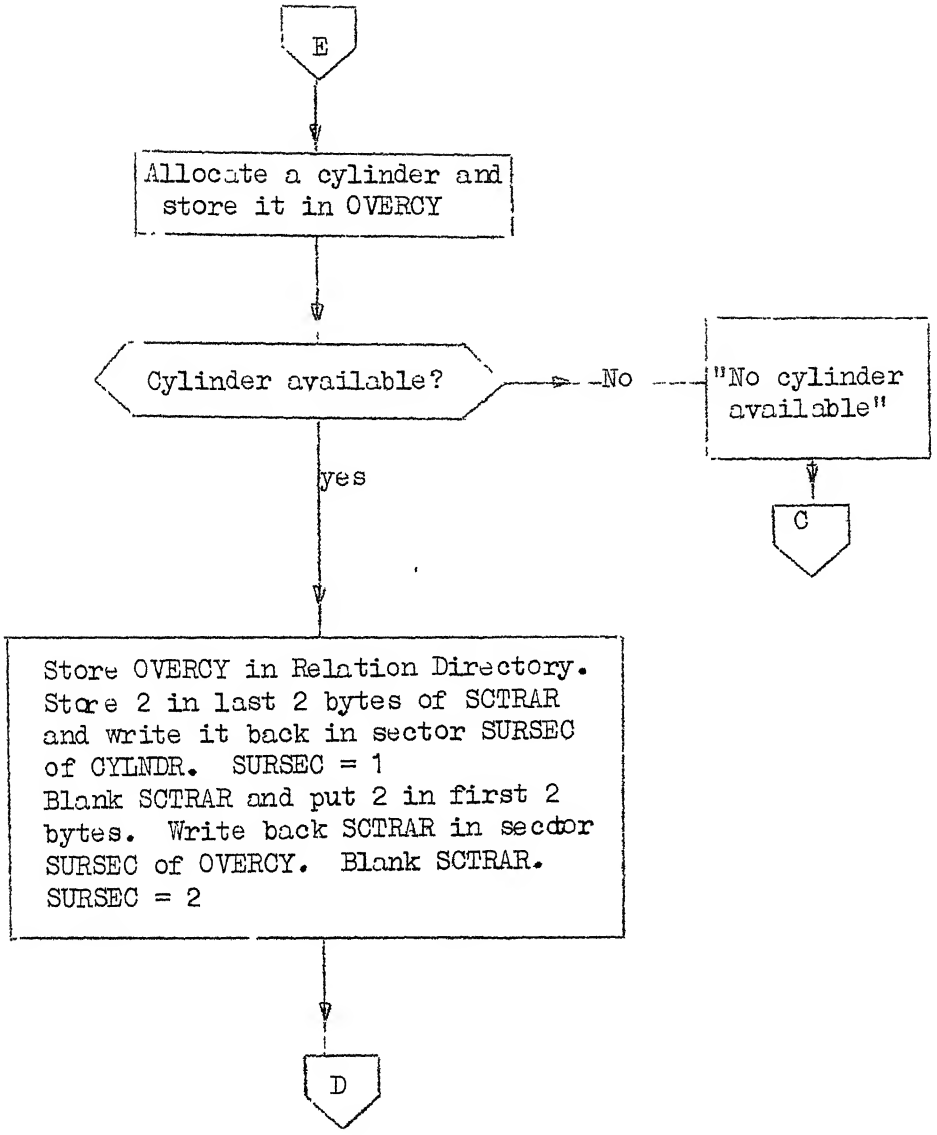
Step 50: Read CYLNDR, NRECSE from primary index table. Call subroutine SUSE to find SURSEC. Is flag MTCHSR = Set? If yes, go to Step 13; else go to Step 30.

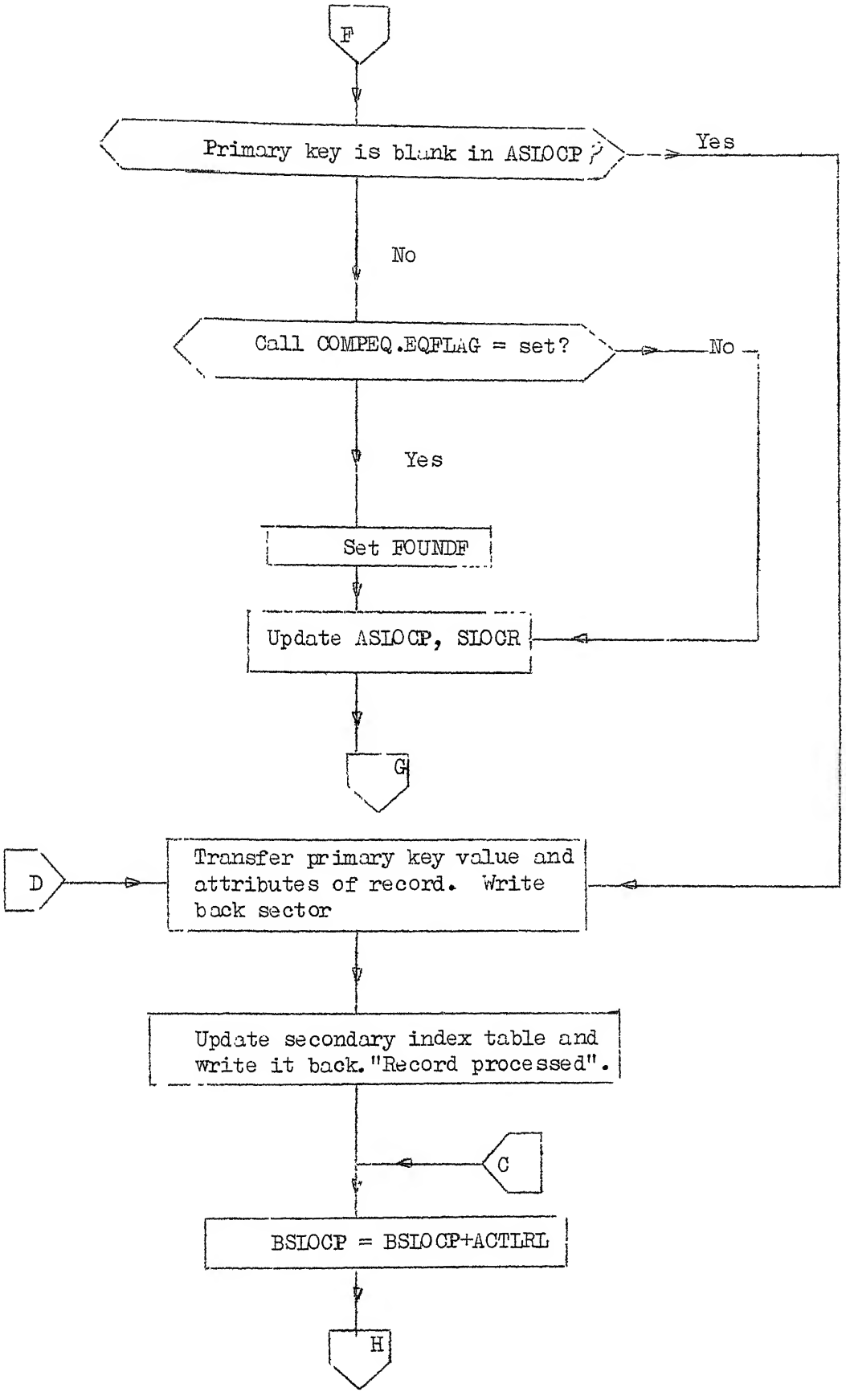
End of Algorithm.

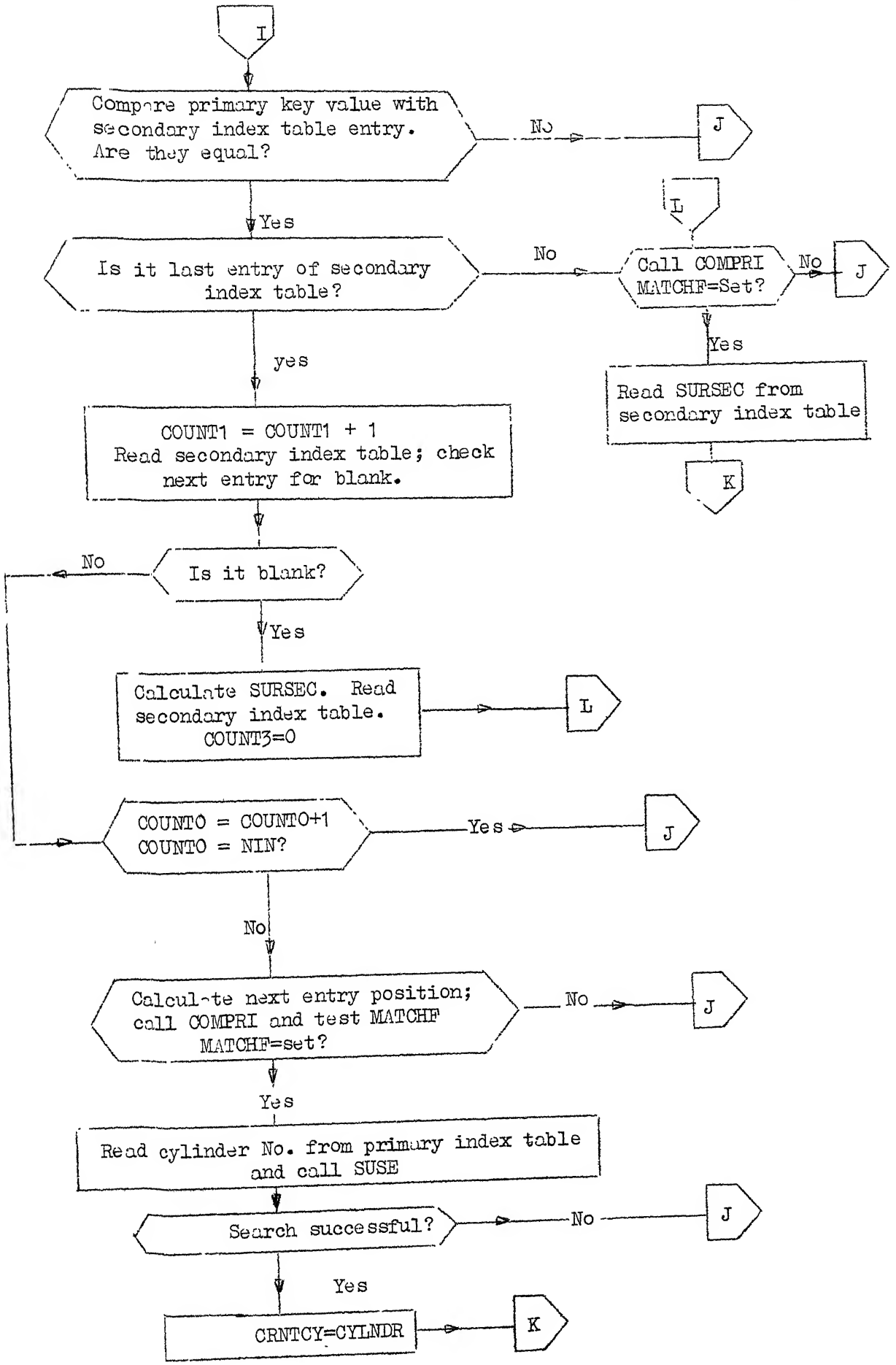
Flowchart for Insertion




A
OVERCY=0?
Yes
No
Read last 2 bytes from SCTRAR in NEWSS
NEWSS = 20040
No
CRNTCY=OVERCY
SURSEC=NEWSS
Set OVERF
Yes
I
Yes
FOUNDF = set?
No
J
Read sector SURSEC from CRNTCY
B
No
OVERCY = 0?
Yes
E
Read 1st sector of OVERCY. Read 1st word and store it in NEWSS
NEWSS = Last sector of cylinder?
Yes
"No space available in overflow cylinder"
C
No
Call NXTSEC and store in NEWSS
Store NEWSS in 1st word and write it back in first sector of OVERCY. Read sector SURSEC from cylinder CRNTCY and store NEWSS in last 2 bytes and write it back.
SURSEC = NEWSS
Make SCTRAR blank.
D


E
Allocate a cylinder and store it in OVERCY
Cylinder available?
No
"No cylinder available"
C
yes
Store OVERCY in Relation Directory. Store 2 in last 2 bytes of SCTRAR and write it back in sector SURSEC of CYLNDR. SURSEC = 1
Blank SCTRAR and put 2 in first 2 bytes. Write back SCTRAR in sector SURSEC of OVERCY. Blank SCTRAR. SURSEC = 2
D


F
Primary key is blank in ASIOCP ?
Yes
No
Call COMPEQ.EQFLAG = set?
No
Yes
Set FOUNDF
Update ASIOCP, SIOCR
G
D
Transfer primary key value and attributes of record. Write back sector
Update secondary index table and write it back. "Record processed".
C
BSIOCP = BSIOCP+ACTLRL
H


I
Compare primary key value with secondary index table entry. Are they equal?
No
J
Yes
Is it last entry of secondary index table?
No
L
Call COMPRI MATCHF=Set?
No
J
Yes
Read SURSEC from secondary index table
K
yes
COUNT1 = COUNT1 + 1
Read secondary index table; check next entry for blank.
No
Is it blank?
Yes
Calculate SURSEC. Read secondary index table.
COUNT3=0
L
COUNTO = COUNTO+1
COUNTO = NIN?
Yes
J
No
Calculate next entry position; call COMPRI and test MATCHF
MATCHF=set?
No
J
Yes
Read cylinder No. from primary index table and call SUSE
Search successful?
No
J
Yes
CRNTCY=CYLNDR
K

8. RESULTS and DISCUSSIONS

For updating operation, SCAN subroutine and all the program subroutines described so far are to be loaded into memory. SCAN occupies memory location from 30000 to 46234 and rest of the program, subroutines occupy memory location from 54000 to 110500. After loading, execution is to be started from address 54000. Then execution of Phase-I is over, control automatically transfers to Phase-II.

Typical examples of different messages received in keyboard while updating program is being executed are given on the next page.

Discussion

Algorithms written for modification and insertion operations have the following limitations:

In single-relation modification, all the fields which are common between specified relation and user-file are being updated in the specified relation only. If these fields exist in any relation other than the specified one, this will lead to discrepancy in data base. So before processing single-relation modification operation, it should be checked from FDLIST that such a situation does not arise. However, if such a situation arises, processing should not stop but it should automatically transfer to group-relation modification operation.

On the other hand, in group-relation modification, it may happen that a field specified in user file exists in more than one relation having different primary keys. According to the algorithm for group-relation modification this field will be modified only in those relations

```
OPERATION STARTS, SPECIFY TYPE OF JOB
FOR UPDATING PRESS U
FOR RETRIEVAL PRESS R
FOR DBA PRESS D
THEN GIVE LINE FEEDU
                    GIVE IDENTIFICATION CODE
ABC

GIVE INPUT MEDIUM
FOR CARD READER PRESS C
FOR KEYBOARD PRESS K
THEN GIVE LINE FEEDC

SPECIFY USER FILE WITH PRIMARY KEY AS FIRST ENTRY


FOLLOWING TABLE PRINTS USER FILE
FIRST COLUMN GIVES FIELD CODE.
SECOND COLUMN GIVES FIELD LENGTH
THIRD COLUMN GIVES DISTANCE IN DATA CARD
FOURTH COLUMN GIVES DISTANCE AFTER REMOVING FILLERS.
FIFTH COLUMN GIVES FORMAT CODE.

F      1              8.          0.          0.          030010
F     45              8.         60.          8.          030010
F     25              6.        108.         16.          030006

IS IT O.K.?

SPECIFY INPUT MEDIUM FOR DATA
C
 CARD-READER READY?

RECORD NO.     1 ., ACCEPTED AND STORED
RECORD NO.     2 ., ACCEPTED AND STORED
RECORD NO.     3.,  ACCEPTED AND STORED
RECORD NO.     4 ., ACCEPTED AND STORED
RECORD NO.     5 ., ACCEPTED AND STORED
RECORD NO.     6.,  ACCEPTED AND STORED
END OF DATA
NO. OF RECORDS IN USER FILE      6.
        **PHASE-I OVER**
SPECIFY TYPE OF UPDATING
CODE FOR DELET=D,INSERT=1,MODIFY=M
I
  SPECIFY WHETHER UPDATING IS ON SINGLE RELATION OR GROUP OF RELATIONS
SINGLE=S,     GROUP=G
S
 SPECIFY RELATION CODE
4
 ALL SPECIFIED FIELDS ARE NOT PRESENT IN GIVEN RELATION
SHALL I PROCEED?

RECORD NO.     1 ., PROCESSED
RECORD NO.     2 ., PROCESSED
RECORD NO.     3 ., PROCESSED
RECORD NO.     4.,  PROCESSED
RECORD NO.     5 ., PROCESSED
RECORD NO.     6 .. PROCESSED
```

Example-2

```
OPERATION STARTS, SPECIFY TYPE OF JOB
FOR UPDATING PRESS U
FOR RETRIEVAL PRESS R
FOR DBA PRESS D
THEN GIVE LINE FEEDU
                    GIVE IDENTIFICATION CODE
ABC

GIVE INPUT MEDIUM
FOR CARD READER PRESS C
FOR KEYBOARD PRESS K
THEN GIVE LINE FEEDK

SPECIFY USER FILE WITH PRIMARY KEY AS FIRST ENTRY
22   PNUM F1 X*8  .

00   .


FOLLOWING TABLE PRINTS USER FILE
FIRST COLUMN GIVES FIELD CODE.
SECOND COLUMN GIVES FIELD LENGTH
THIRD COLUMN GIVES DISTANCE IN DATA CARD
FOURTH COLUMN GIVES DISTANCE AFTER REMOVING FILLERS.
FIFTH COLUMN GIVES FORMAT CODE

F      1              8.            8.            0.          030010

IS IT O.K.?

SPECIFY INPUT MEDIUM FOR DATA
K
 GIVE INPUT DATA FROM KEY-BOARD
IC 00030
RECORD NO.     1 ., ACCEPTED AND STORED
IC 00040
RECORD NO.     2 ., ACCEPTED AND STORED

END OF DATA
NO. OF RECORDS IN USER FILE       2 .
        **PHASE-I OVER**
SPECIFY TYPE OF UPDATING
CODE FOR DELET=D,INSERT-I,MODIFY=M
D
 SPECIFY WHETHER UPDATING IS ON SINGLE RELATION OR GROUP OF RELATIONS
SINGLE=S,    GROUP=G
G
 *****RELATION NO.      4
RECORD NO.     1 ., RECORD DOES NOT EXIST IN SECTOR
RECORD NO.     2 ., PROCESSED
 *****RELATION NO.      6
RECORD NO.     1 ., RECORD DOES NOT EXIST IN SECTOR
RECORD NO.     2 ., RECORD DOES NOT EXIST IN SECTOR
```

whose primary key is the same as primary key of user file. This may again lead to discrepancy in data base. Hence after a group-relation modification is over, it should be checked from FDLIST whether such a situation arises and if it does, processing should not stop but it should repeat itself and ask for a user file with different primary key.

While inserting a record it is never being checked whether the field-values inserted in a relation for a given primary key are consistent with field values for the same primary key in the same relation or other relation. As a result, it may sometime happen that for the same primary key there may be records with different field-values for a single-valued field, creating discrepancy in data base. This can be avoided if all these field values for given primary keys are retrieved from data base before starting insertion operation and manually verifying that there is no inconsistency between retrieved information and informations going to be inserted. This verification can also be done by writing a program.

Removing all these limitations of Modify and Insert algorithms and making these algorithms more flexible and more realistic may be a further extension of this work.

LIST OF REFERENCES

1. Chamberlin, D.D., "Relational Data Base Management Systems", ACM Computing Surveys, March 1976, Vol. 8, No. 1, pp. 43-66.
2. Codd, E.F., "A Relational Model of Data for Large Shared Data-Banks", Comm. of ACM, Vol. 13, No. 6, June 1970, pp. 377-397.
3. Date, C.J., "An Introduction to Data Base Systems", Addison-Wesley Publishing Company, 1975.
4. Ghanekar, D.K., "An Implementation of a Relational Data Base Model", Computer Science Program, IIT Kanpur, August 1977. (M.Tech. Thesis).
5. Martin, J.,"Principles of Data Base Management", Prentice-Hall of India Pvt. Ltd., New Delhi, 1977.
6. Rajaraman, V., "Design of Information Systems", Class Notes of the Course CS 650, Computer Science Program, IIT Kanpur, January 1977.
7. Salton, G., "Automatic Information Organization and Retrieval", McGraw-Hill Book Company, 1968.
8. Sharma, R.K., "A Personnel Data Base Retrieval Systems", M.Tech. Thesis, Computer Science Program, IIT Kanpur, July 1977.
9. Sibley, E.H., Fry, James, P., "Evolution of Data-Base Management Systems", ACM Computing Surveys, March 1976, Vol. 8, No. 1, pp.7-72.
10. Widerhold, G., "Data Base Design", McGraw-Hill Book Company, 1977.